YEDITEPE UNIVERSITY
FACULTY OF ENGINEERING

Determination of Ochratoxin in Raisins Through NIR Scan with Artificial Intelligence Based Methods

Ali Baki TÜRKÖZ

GRADUATION PROJECT REPORT
Department of Electrical and Electronics Engineering

Supervisor
Asst. Prof. Dr. İpek BAZ

ISTANBUL, 2024

YEDITEPE UNIVERSITY
FACULTY OF ENGINEERING

Determination of Ochratoxin in Raisin Through NIR Scan with Artificial Intelligence Based Methods

by

Ali Baki TÜRKÖZ

May 24, 2024, Istanbul

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

YEDITEPE UNIVERSITY

Signature of Author(s) ……

Department of Electrical and Electronics Engineering

Certified By ……

Project Supervisor, Department of Electrical and Electronics Engineering

Accepted By ……

Head of the Department of Electrical and Electronics Engineering

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to everyone who supported and contributed to the realization of this project.

Firstly, I am deeply grateful to my project advisor, Assist. Prof. Dr. İpek BAZ, for her guidance throughout my undergraduate education and for her valuable support and advice during this project. Her vast knowledge and guidance were a great source of inspiration at every stage of the project.

I would also like to thank Yeditepe University for providing the necessary equipment and resources for the completion of my project. The facilities offered by our university played a significant role in the success of my work.

My deepest gratitude goes to my beloved family, especially my mother Gülşen TÜRKÖZ, my father Baki TÜRKÖZ, and my sister Fatma TÜRKÖZ, for their unwavering belief in me and their exceptional support throughout my educational journey. My family has always been my greatest source of motivation.

I once again thank everyone who contributed to this project.

May, 2024

Ali Baki TÜRKÖZ

TABLE OF CONTENTS

Table of Contents

ABSTRACT

Grapes, or Vitis vinifera as they are scientifically known, are a very popular type of fruit. However, the problem of mycotoxin contamination of grapes and grape products has become a serious health threat. One of these mycotoxins is Ochratoxin A (OTA), which is particularly prevalent in grapes and grape products. OTA is a type of mycotoxin known to have carcinogenic, teratogenic, immunotoxic and neurotoxic effects for humans. It is also reported to cause kidney diseases and cancer. It is classified as a probable human carcinogen by the International Agency for Research on Cancer (IARC) [1].

Türkiye is crucial since it plays a significant role in raisins' manufacturing and exportation of raisins. *Table 1.1* and Table 1.2 show Turkey's raisin data and Turkey's position in the World Raisin industry. Although, the grape industry has been grappling with OTA contaminations in the previous years. This was potentially dangerous to human beings and Turkish raisin business as well. It is essential that OTA be detected and controlled in raisins [2].

This study has two goals:

1. Data Collection on Mycotoxins via NIRscan Nano: Using NIR scan Nano spectroscopy, this research will collect information from healthy and mycotoxin-contaminated raisins. Thus, an artificial intelligence model for mycotoxin detection can be developed and tested using these data.

2. Autonomous Detection System based on Artificial Intelligence (AI): An AI-based self-directed system will be created to identify mycotoxins in raisins. It employs NIR Scan Nano technology to mechanically differentiate between raisins contaminated with mycotoxins.

Therefore, if it is possible to detect OTA through this system that utilizes the NIRscanNano technology, it means that Türkiye's market for raisin will have a competitive advantage over others because it will provide better health wise products.

In conclusion, this paper aims at protecting human life as well as contributing to the world's raisin industry through providing an innovative approach of detecting OTA mycotoxin. The artificial intelligence-based detection system facilitates fast but efficient detection of mycotoxins. By preventing toxicants harmful to human health from entering global supply chains at the earliest stage and thus saving many raisons going to waste beside its other contribution toward human health which goes beyond just ensuring prevention of toxicants harmful to human health from infiltrating global supply chain at their source.

Keywords: Artificial Intelligence, Deep Learning, NIR scan Nano, Autonomous, Human Health

.

LIST OF SYMBOLS

σ: Sigmoid

Σ: Upper Letter Sigma

ABBREVIATIONS

CNN: Convolutional Neural Network

OTA: Ochratoxin

NIR: Near-Infrared

MIR: Mid-infrared

FIR: Far-Infrared

EVM: Evaluation Module

Nano: Nanotechnology

R&D: Research and Development

IARC: International Agency for Research on Cancer

LC: Liquid Chromatography

HPLC: High-Performance Liquid Chromatography

FL: Fluorescence

MS: Mass Spectrometry

GC: Gas Chromatography

DRBC: Dichloran Rose Bengal Chloramphenicol

CFU: Colony Forming Unit

FB2: Fumonisin B2

IA: Immunoaffinity

EU: Europe

FLD: Fluorescence Detection

TLC: Thin-Layer Chromatography

EIA: Enzyme Immunoassay

ELISA: Enzyme-Linked Immunosorbent Assay

LIST OF FIGURES

LIST OF EQUATIONS

LIST OF TABLES

1. INTRODUCTION

Raisins are among the fruits sensitive to fungal development, and according to studies conducted in this field, ochratoxin A (OTA) has been identified as the most encountered mycotoxin in raisins and grape products. Considering that individuals, especially children, consume large amounts of raisins, raisins varieties are thought to be a significant source of OTA intake.

Ochratoxin A (OTA) is a mycotoxin. This toxin is carcinogenic to rodents. It is teratogenic, immunotoxic, and likely neurotoxic and genotoxic. Additionally, it has been shown to be the cause of urinary tract tumors in humans. It leads to kidney diseases and cancer in humans. Ochratoxin A was classified as a possible human carcinogen (Group 2B) by the International Agency for Research on Cancer (IARC) in 1993 [1].

Türkiye üzüm verileri (1.000 ton) / Türkiye grape datas (1,000 tons)

	2016/2017	2017/2018	2018/2019	2019/2020	2020/2021
Alan/Area (bin da)	4.169	4.170	4.054	4.010	3.902
Verim/Yield (kg/da)	1.007	943	1.011	1.050	941
Üretim/Production	4.000	4.200	3.933	4.100	4.208
Tüketim/Consumption	2.288	2.186	2.181	2.363	2.468
Stok değişimi/Stock change	-82	39	-20	-19	-12
İthalat/Imports	8	9	131	11	146
İhracat/Exports	1.231	1.397	1.339	1.197	1.261

Table 1.1: Türkiye Raisin Datas (tarimorman.gov.tr)

Dünya Üzüm verileri (1.000 ton) / World grape data (1,000 tons)

	2016	2017	2018	2019	2020
Alan/Area (1.000 ha)	6.900	6.833	6.873	6.912	6.951
Verim/Yield (ton/da)	1.079	1.076	1.165	1.114	1.123
Üretim/Production	74.436	73.516	80.044	77.000	78.034
Yaş üzüm ithalat/Fresh grapes imports	4.207	4.560	4.525	4.718	4.624
Yaş üzüm ihracat/Fresh grapes exports	4.555	4.87	4.798	4.909	4.690
Kuru üzüm ithalat/Dried grapes imports	897	877	898	864	785
Kuru üzüm ihracat/Dried grapes exports	903	874	915	872	808
Fiyat/Price ($/kg)	1,98	1,79	1,98	2,17	2,03

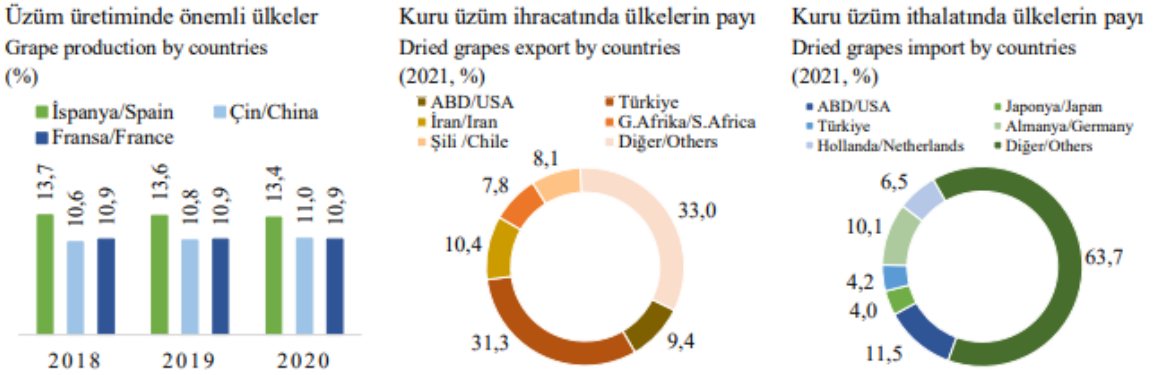

Table 1.2: World Raisin Datas (tarimorman.gov.tr)

Türkiye ranks among the top countries globally in terms of raisin exports. According to data from the Anadolu Agency in 2022, Türkiye exported 252 thousand 354 metric tons of raisins, amounting to a total export value of $441,862,000 for the period spanning September 1, 2021, to August 31, 2022. Our country accounts for 33% of global exports in seedless raisins.

Mehmet Ali Işık, President of the Aegean Dried Fruit and Products Exporters' Association, stated that Türkiye alone covered 22% of the global production, which amounted to approximately a million 301 thousand tons, with a harvest of around 290 thousand tons during the 2021-2022 season [3].

In the Agriculture Product Markets article published by TCTOB in July 2023, it was stated that Türkiye meets 31.3% of the global raisin export in 2022, with our country having realized 931 thousand tons of raisin exports in 2021, and the domestic consumption of seedless raisins in Türkiye being at the level of 35-50 thousand tons.

Raisins, which are an important export product for Türkiye, faced a significant problem due to the presence of OTA revealed in analyses conducted in the UK for the products of the 1996-1997 season, leading to a major OTA crisis in seedless raisins exported in 1997 [2].

It is important to conduct periodic screening studies for the formation of mycotoxins in dried fruits, which make a significant contribution to the country's economy, to ensure the continued and increased demand by foreign countries, and to identify and solve problems in advance. It is believed that mycotoxins can also be removed after contamination through conscientious practices and good technology [4].

This study is aimed at detecting mycotoxins in raisins by employing an Artificial Intelligence-based autonomous detection system and improving product quality to reduce health risks. So far, no data has been recorded regarding OTA detection in raisins using hyperspectral imaging techniques. The current methods of OTA detection employ chemical analyses, which cannot be used on individual products since millions of them have to be inspected every hour. For this reason, it is possible for the producers of raisins to come up with more dependable ones if an AI-based detector identifies the mycotoxins in real time and with a high sensitivity.

The primary goal of this research is to make easy and automatic the detection process for highly hazardous substance OTA that can also be applied for other things such as Raisin.

Ochratoxin A (OTA), a fungus belonging to a group known as molds, usually enters soil before finally settling on raisins through surface cracks where it multiplies. In another study conducted in Italy, Aspergillus and Penicillium species were isolated from raisins, and it was noted that temperature, rainfall, and humidity influenced the formation of mycotoxins. Research findings indicate that even in samples where mold is not detected, OTA levels are measured, suggesting that although the mold may not persist in the environment after toxin production, the toxin may remain in the product [4]. As a result of the detection of these highly harmful substances, products exported are sent back to Türkiye for destruction, further exacerbating our losses.

Currently, chemical, and immunological methods such as HPLC and ELISA are used for the detection of OTA. These methods are inefficient both in terms of time and economy. Additionally, they are inadequate due to both the inadequacy and unreliability of being a

sample test, where a certain amount represents the entire batch, and the wastage of the processed quantity.

In our country, the maximum limit value for OTA in dried raisin samples is set at 10 µg/kg according to the Türkiye Food Codex Regulation on Contaminants. Mold limits are evaluated according to the sampling plan specified in the Türkiye Food Codex Microbiological Criteria Regulation. The maximum mold limit value found in individually collected dried raisin samples is 10^5 CFU/g. [4].

The aim of this project is to overcome the limitations of chemical and immunological methods by developing a faster and more reliable detection method using NIR Spectrometer and artificial intelligence. The current research is focused on developing an automated OTA detection system based on artificial intelligence. The NIR Spectrometer will be employed as a virtual tool to sense the presence of mycotoxins in grapes and predict their formation earlier in advance so that conditions for treating them at their initial stages become possible. Hence, this study aims to introduce a new approach to detect mycotoxin contamination in raisins with minimum health hazards and high-quality finished products. The detection system relies on AI to give out any detected levels of mycotoxins as they happen, thus making it possible for producers of raisin to depend on a better product.

1.1. Thesis Content

There are seven chapters in this thesis.

Chapter One (Introduction): This part gives an introduction of the work and highlights the current motivation. It discusses how OTA is common in raisins and grape products with emphasis on health risks associated with consumption of such products. Besides, it focuses on the raisin industry in Turkey, which is affected by OTA contamination.

Chapter Two (Research Objective): The research objectives that delve into deep learning for detecting OTA in raisins are found in section two. This gives a detailed description of what deep learning concepts are and what convolutional neural network layers entail. It also includes specifications of designs and parameters for the project.

Chapter Three (Literature Review): A comprehensive literature review is carried out including information about previous studies on OTA detection, mycotoxin contamination in raisins among others. In this section, readers get to know the model and methodology used in different recent studies.

Chapter Four (Design): The design segment talks about practical considerations and limitations involved within the project. It outlines how the project was done as well as describing a workflow followed during detection of OTA in raisins.

Fifth Chapter (Methods): This section includes a detailed explanation of the methods that is used in this project. It explains which technologies such as artificial intelligence and NIR Spectrometry are used and how they are used. Also, this chapter includes a description of how algorithms developed for OTA detection operate.

Sixth Chapter (Results and Discussion): This chapter includes the results and findings. Also, the performance of the developed models is analyzed and the findings the results are evaluated.

Seventh Chapter (Conclusion): In this chapter, the importance of this research is mentioned and also the recommendations for further studies balanced on.

2. RESEARCH OBJECTIVE

The impact of infrared frequencies on substances and the rationale for the use of Near Infrared (NIR) spectroscopy are significant components of this study. Understanding these factors enables the applicability of NIR spectroscopy in the detection of ochratoxin A in raisins.

Effect of Infrared Frequencies on Substances

$0.7 < \lambda < 1000$ μm is the region where infrared radiation exists, which lies between visible light and microwaves. This range consists of near, mid and far-infrared regions. The absorption or emission of energy through molecular vibrations is mainly excited by infrared interaction with substances. Unique spectral fingerprints are developed as different substances undergo certain frequencies of infrared radiation absorption.

- Near Infrared (NIR) Region (0.7-2.5 μm): -Near Infrared (NIR) Region (0.7-2.5 μm): Overtones and combination vibrations for molecules are particularly important within this region because it offers information on molecular composition/structure for organic compounds such as water, fats, proteins, and carbohydrates NIR penetrates samples deeper than both mid-IR and far-IR enabling nondestructive analysis [1].

- Mid Infrared (MIR) Region (2.5-25 μm): It is used for fundamental molecular vibrations and provides valuable data on the structure of molecules that are measured. However, this technique typically requires a greater amount of sample preparation and can be more destructive [2].

- Far Infrared (FIR) Region (25-1000 μm): This region is less commonly used for organic substances because its low energy is less effective in inducing molecular vibrations associated with organic compounds [3].

Reasoning for the choice of near infrared spectroscopy

Near Infrared (NIR) spectroscopy has been chosen for several reasons:

1. Non-Destructive Analysis: NIR spectroscopy allows analysis of samples without altering or destroying them, which is critical for preserving the integrity of samples in the detection of ochratoxin A in raisins [1].

2. Speed and Efficiency: NIR spectroscopy is highly appropriate for regular screening processes since it is very fast in terms of analysis and data collection; this aspect enables timely detection of contaminants in different food products. At the same time, this speed of NIR analysis makes it ideal for quick screening processes involved in quality control procedures [2].

3. Depth of Penetration: Samples are penetrated by near-infrared (NIR) to a greater depth than in other regions This indicates that this region of the spectrum does not just look at surface properties but also at internal structures and bulk characteristics of samples too. Consequently, NIR spectroscopy is crucial for detecting such contaminants present in foods unevenly distributed around them with ochratoxin A being one such contaminant. It can therefore be used for non-invasive analysis hence offering a more comprehensive and consistent sample assessment [3].

4. Minimal Sample Preparation: The analysis process time and cost are reduced by minimal, or no sample preparation requirements associated with most NIR spectroscopy approaches [1].

5. Sensitivity to Organic Compounds: The NIR spectroscopic technique has been developed as an analytical tool for detecting mycotoxins like ochratoxin A because it shows good correlation between certain molecular vibrations specific to each common organic compound and absorbance at different wavelength bands within its region [2].

For this study, it is chosen for Near Infra-Red spectroscopy mainly because of its non-destructive nature, quick analyzing capabilities, deep penetration ability, minimal sample requirements and being highly sensitive to organics. These strengths thus make NIR spectroscopy an excellent method for the detection of ochratoxin A in raisins.

In this project, the Texas Instruments DLP NIRscan Nano Evaluation Module Spectrometer was used. The internal structure of this device is shown in *Figure 2.1*. Spectral data were collected at wavelengths ranging from 900 to 1700 nm, thus investigating the visibility of OTA at different wavelengths.

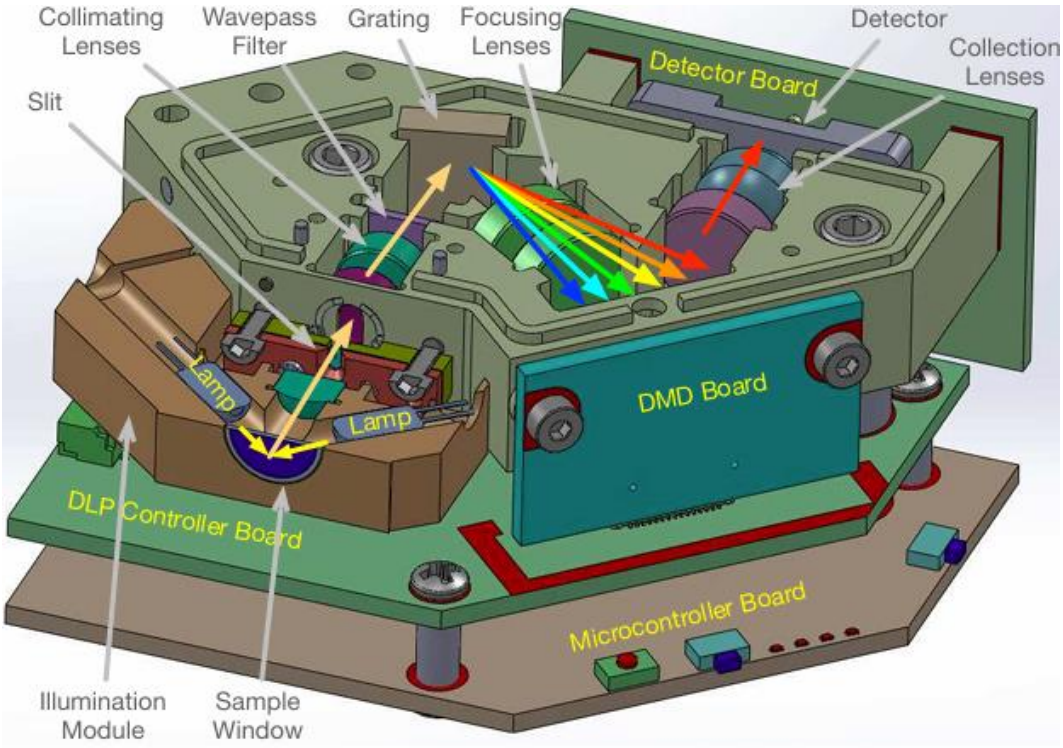

Figure 2.1:DLP NIRscan Nano Optical Layout(www.ti.com)

A spectrometer is a type of camera which can capture spectral information or images that are not visible to the human eye. There are different types of spectrometers, including near-infrared (NIR), mid-infrared (MIR), and far infrared (FIR), which can detect radiation at different wavelengths or spectral bands. Therefore, the decision has been made to use spectral cameras. Convolutional neural network-based models have been trained for classification with spectral images and utilized for the detection of OTA in raisins.

2.1. Convolutional Neural Networks

CNNs or Convolutional neural networks are types of artificial neural network models which analyze pictures and learn the difference about it. CNNs. CNNs are artificial neural network models that detect changes by applying convolution on images and calculating gradients in images and learn images by correlating these changes with each other correctly. It is also called Deep Learning or Deep Neural Networks (ANNs) since it has many layers of nodes that transform input data into final output data. CNNs has reduced the number of parameters on ANNs which is one of their greatest advantages [6]. This achievement has prompted researchers and developers to work with more complex assignments using larger models, unlike classical ANNs. The most used deep neural networks are Convolutional Neural

Networks (CNN). It has the mathematical linear operation of convolution between matrices [1]. The matrix-based filters applied on data in image processing studies have emphasized features relevant to image classification. For instance, it is possible to independently detect faces in a face detection application irrespective of their location. Another feature is that as we go deeper into the layers, we start getting abstract features. For instance, in the initial layers, edges and corners can be detected, in the subsequent layers, simpler shapes, and as we progress to further layers, more detailed information such as color pattern changes, molecular changes, or facial expression changes depending on the emotional state on the face can be detected. There are lots of CNN models such as AlexNet, VGG, GoogLeNet, ResNet, EfficientNet, and MobileNet in the literature.

2.2. Layers of a Convolutional Neural Network

CNN consists of multiple layers, including convolutional layer, non-linearity layer, pooling layer, and fully connected layer. Convolutional and fully connected layers have parameters, but pooling and non-linearity layers do not. CNN demonstrates excellent performance in machine learning problems. Particularly in applications dealing with image data, results obtained in fields such as computer vision and natural language processing (NLP) have been very impressive, with the largest image classification dataset (ImageNet) being a notable example [1].

2.2.1. Convolutional Neural Layer

A convolutional neural network (CNN) is a deep neural network that identifies patterns in images. In the matter of signal processing, computer vision and pattern recognition, these neural networks carry out several operations. In terms of the human brain, it is like a network of neurons, but in their convolutional networks this structure is mirrored. Convolution is a procedure that takes one function and through it produces another. CNNs arrange local connections of neurons in adjacent layers and employ filters to test if some graphic images may contain certain features present in all graphics i.e., changes related to different wavelengths. This allows for the extraction of features that are imperceptible to the human eye and the detection of changes. Among the fundamental functions of convolutional neural

networks are nonlinear convolution dynamics (ReLU), fully connected layers, and pooling or subsampling categorization. These networks typically start with a convolutional layer before processing the input and perform various operations before sending it to the next layer. Thus, the convolution technique is used to reduce the number of pixels in an image. With each reduction process, the depth of the image increases, allowing for the extraction of deeper and more abstract information.

The success of deep convolutional networks is often based on two main convolution properties: translation invariance and locality. The locality property reduces the number of parameters and FLOPs (number of addition and multiplication operations), and in recent years, different types of convolutional layers such as k×k spatial convolution filters, 1×1 pointwise convolution filters, separable convolutions (SCs), and depthwise separable convolutions (DSCs) have been used. These layers have been used in many ConvNets models, indicating a focus on improving the performance of computer and mobile embedded ConvNets on image datasets [2].

Adding more neurons to the hidden layer increases the total number of parameters, resulting in a substantial increase in complexity. For example, if we were to add more neurons, say doubling them, it would lead to a significant increase in parameters, from 32x32x3x2 to 32x32x3x4, which is a considerable load. To illustrate, just connecting two nodes would require over 6000 weight parameters, indicating the high demand for resources.

When considering applications like image classification, it might seem that two neurons in a hidden layer wouldn't suffice due to the complexity of image data. To address this, one might think of connecting the input image to neurons in the subsequent layer with the same dimensions, akin to operations like edge detection in images.

However, this approach introduces many connections. This demonstrates the considerable computational burden involved [1].

Therefore, the pursuit of a more efficient method seems to be looking at local regions in the image instead of the whole image [1].

Although the size of the connections greatly decreases, there are still many parameters to be resolved. Another assumption for simplification is to keep the local connection weights the same for all neurons in the next layer. This connects neighboring neurons in the next layer to the local region of the previous layer with exactly the same weight. Thus, it reduces many extra parameters again [1].

There are many benefits to these simple assumptions. Firstly, the number of connections decreases. Secondly, a more interesting concept is that fixing the weights of local connections is like sliding a window on the input neurons and mapping the produced output to the relevant place. This provides an opportunity to detect and recognize features at any location in the image. Therefore, they are called convolutions [1].

Matrix can be adjusted to detect edges in the image. Since these matrices act like classical filters, they are also called filters. However, in convolutional neural networks, these filters are initialized, and then the training procedure continues with shape filters more suitable for the given task [1].

To make this method more useful, it is possible to add more layers after the input layer. Each layer can be associated with different filters. Therefore, we can extract different features from the given image [1].

2.2.2. Pooling Layer

The pooling layer performs subsampling to reduce complexity in subsequent layers. For this process, a kernel size and stride are first determined. Then, a single output is generated for all pixels covered by the specified kernel size by applying a filter, and then the filter is shifted by the specified stride on the feature map extracted by the previous Convolutional layer, thus passing the entire feature map through the filter. If padding operation is not applied, the

output produced by the pooling operation will be at a lower size than the original image dimensions. This operation is akin to reducing resolution in image processing. Pooling does not affect the number of filters. There are different pooling methods. Max-pooling is one of the most common pooling methods. In max pooling, the maximum value within the selected kernel is chosen and recorded as a single value. The selected kernel is shifted over the original image by the length of the selected step, allowing for the creation of a new image. The larger the selected kernel size, the smaller the dimensions of the resulting new image will be, as the pixels within the filter are reduced to a single representative value. Before performing pooling layer activities, it is important to understand how to select a pixel pool and obtain a representative value. Usually, a square matrix with a variable number of inputs is used to select adjacent pixels. The average or maximum of the selected pixels often serves as the representative value.

One of the most used sizes in max-pooling is 2×2. For example, as seen in the *Figure 2.2*, when pooling is performed in the lower-left 2×2 blocks (pink area), it moves 2 and focuses on the lower-right (blue area). This means that the pooling step (stride) is 2. If no subsampling is to be performed, a rarely used step of 1 can be used. However, it is important to remember that information location is not preserved. Therefore, information presence (when it matters) should be considered over spatial information. Additionally, pooling can be used with unequal filters and steps to increase efficiency. For example, a 3x3 maximum pooling with a step of 2 maintains some overlaps between areas [1].

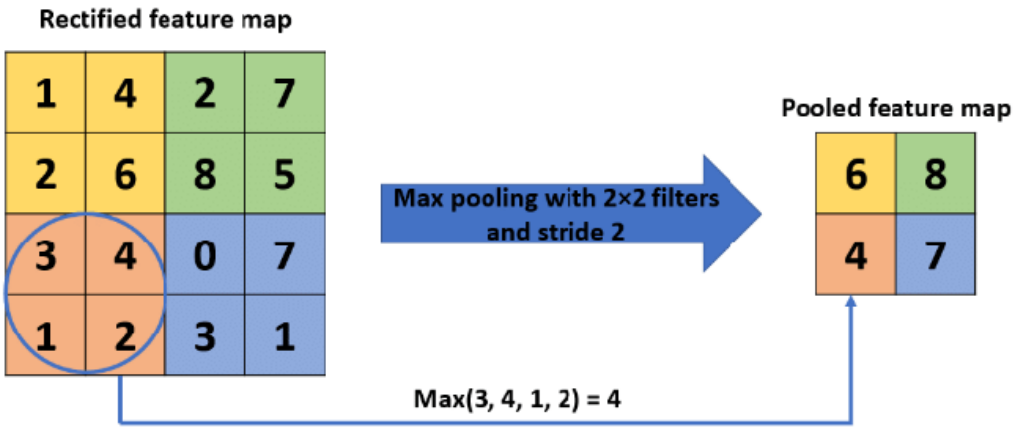

Figure 2.2: Maxpooling Operation (Researchgate.net)

2.2.3. Batch Normalization

Training deep neural networks is a complex and challenging process due to the changes in the distributions of inputs at each layer during training. If training is attempted with complex and irregular input data, low learning rates may be required, and careful parameter tuning may be necessary, resulting in slow training, particularly for non-linear models. This phenomenon is referred to as internal covariance shift and is resolved by normalizing the layer inputs. Batch Normalization allows for the use of much higher learning rates and less careful initialization. Additionally, in case of insufficient GPU RAM during training, adjusting the minibatch size contributes significantly to achieving the fastest and most hardware-compatible optimal training level. Moreover, it serves as a regularizer without the need for Dropout in some cases. Thus, it reduces the overall training error, enhances regularization, and accelerates training. By stabilizing the learning process, it significantly reduces the training periods necessary to create deep networks [3].

2.2.4. Dropout

Dropout is a technique that prevents overfitting and effectively combines numerous different neural network architectures. Dropout involves dropping units in a neural network, meaning temporarily removing them from the network along with all their incoming and outgoing connections, as shown in *Figure 2.3*.

The selection of units to be dropped occurs randomly. Each unit is kept with a constant probability, p, independently of other units; here, p can be selected using a validation set or typically set to 0.5, which is often seen as optimal [4].

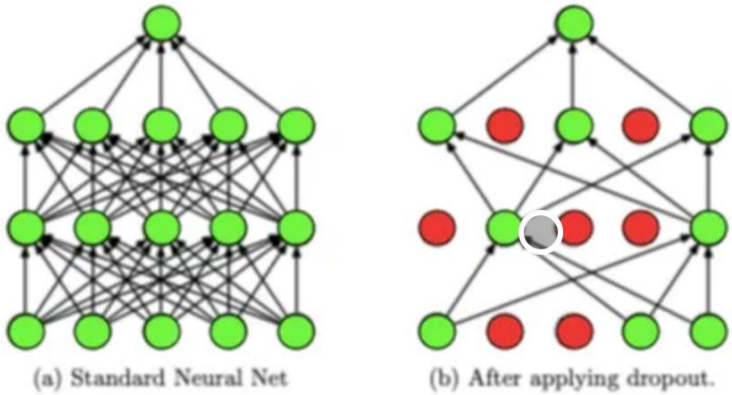

Figure 2.3: Dropout Operation

2.2.5. Fully Connected Layer

Fully connected layers constitute the last few layers of the network, simply representing feedforward neural networks. The input data to a fully connected layer are the flattened outputs of the last Pooling or Convolutional Layer. The output from any Pooling or Convolutional Layer is a 3-dimensional matrix; flattening this matrix converts all its values into a vector. This flattened vector is then connected to several fully connected layers; these layers are identical to Artificial Neural Networks and perform the same mathematical operations *Equation 2.1*.

For each layer of the Artificial Neural Network, the following calculations are made:

$$g(Wx + b)$$

Equation 2.1: Activation Level of the Neural Layer

Here,

x => the input vector with dimensions [p_l, 1].

W => the weight matrix with dimensions [p_l, n_l,]; p_l is the number of neurons in the previous layer and n_l is the number of neurons in the current layer.

b => the bias vector with dimensions [p_l, 1].

g => the activation function, typically ReLU.

This calculation is repeated for each layer. After passing through the fully connected layers, the last layer uses the softmax activation function instead of ReLU for classification.

Softmax activation function works by assigning probabilities to each class in terms of how likely it is where these probabilities sum to one for all classes. After that, source data is categorized using its labels.

The formula of *Equation 2.2* can be used to find out what size your output tensor should have:

$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

Equation 2.2: Dimensions of the Output Tensor

Here,

W_1 => the width/height of the input tensor

F => the width/height of the kernel

P => padding

S => stride

W_2 => the width/height of the output

To ensure uniformity of a move of the kernel over an input matrix, padding is commonly employed. When padding is used, a layer of zeros with the given dimensions is added to all borders of the input matrix.

Stride is the distance between two steps. The kernel moves around on pixels in this way: 1-pixel stride. To make the kernel move about 2 pixels per step we can increase stride by 2. It also affects output tensor's dimensions and helps to avoid overfitting.

In such case, for Convolutional Layer, the number of channels in the output tensor equals to number of kernels. And as regards Pooling Layer, channels in both input and output tensors remain constant [5].

2.3. Activation Functions

In deep learning models, activation functions are vital for the processing of inputs. These operate as a gauge for a neuron's output and help in deciding whether to pass any data on to the next piece of information. In deep learning algorithms, it is necessary to correctly select the activation functions so that the algorithm can be said to have a healthy learning procedure, they prevent over-fitting also improve accuracy and reduce computational costs. Another vital role played by such activation functions is that they assist in identifying the hidden characteristics of deep learning models that are useful in different life cases. Nonetheless, selecting the wrong ones may lead to little or no learning or even lack of success.

Parameters are updated using backpropagation algorithm in deep learning algorithms. This process involves updating parameters for each layer by propagating the amount of error at the network's output backwards. Afterward, derivation of activation functions occurs during the update process. Therefore, developers require their used functions for deep neural nets to be differentiable. In deep neural network structures, an activation function is anticipated

to possess certain fundamental attributes such as but not limited to; ability to attain global optima as opposed to local ones, non-linearity, and derivativeness.

It is also important to know how the backpropagation algorithm linked with activation functions. "Backpropagation is an algorithm used to calculate the amount of error in the neural network and then updates parameters by recognizing how this error propagated throughout each layer." This step is very vital to make our model more accurate [6].

2.3.1. Sigmoid

The sigmoid function is preferred for binary classification problems because it transforms values to between 0 and 1. Simplicity and computational speed make the sigmoid function more popular. *Equation 2.3* sigmoid activation function formula makes the calculation simple and easy.

$$f(x) = \frac{1}{1 + e^{-x}}$$

Equation 2.3: Formula of Sigmoid Activation Function

However, when input values get larger or smaller, the slope of the sigmoid function diminishes, and this may cause the vanishing gradient problem. It means that a deep learning network with many hidden layers will have gradients close to zero for most weight updates, which in turn slows down convergence. Thus, this constraint may not be suitable for all classification problems. Therefore, when the input value is very small (bounded at 0) or very large (bounded at 1), as illustrated in *Figure 2.4*, its output will saturate resulting in zero derivative and gradient loss causing information loss. For example, an application of this activation function would be to build a model for detecting ochratoxin in dried grapes as shown in this study. To know whether a particular instance falls under class one, it will calculate their respective probabilities as a real value of 0.0 up to 1.0 to arrive at a decision using Sigmoid function. The sigmoidal equation is also plotted graphically by means of *Figure 2.4* below.

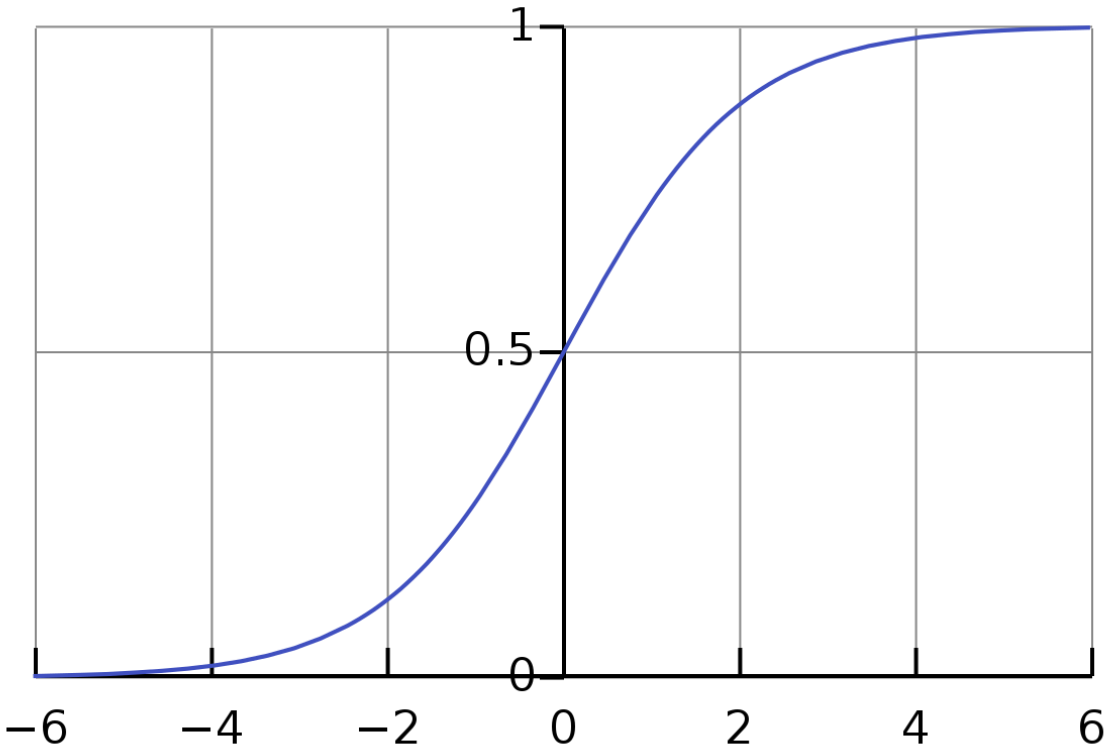

Figure 2.4: Sigmoid Activation Function

2.3.2. ReLu (Rectified Linear Unit)

The ReLU function is a simple and fast computation function as it returns the input value if inut is greater than zero, else returns zero. Moreover, the domain of ReLU function from zero to infinity makes it more useful because this property helps fixing vanishing gradient problem. The $f(x) = \max(0, x) = \ 0\ if\ x < 0 x\ if\ x \geq 0$

Equation *2.4* displays the mathematical representation of ReLu activation Function:

$$f(x) = \max(0, x) = \begin{cases} 0\ if\ x < 0 \\ x\ if\ x \geq 0 \end{cases}$$

Equation 2.4: Formula of the ReLu Activation Function

A more stable learning procedure in deep networks is provided by the ReLU function which works by returning input value if it is greater than zero, otherwise zero. The above notwithstanding, there are cases where ReLUs have been found to be causing dying neurons. When x< 0, dx =0 and thus no change in all that part of the curve with slope equal to one. It's highly likely that such neurons will stop backpropagation. This case shown in *Figure 2.5*

frequently happens in large or deep neural networks and leads to poor performance behavior of models. To avoid such dead neurons, one should consider these issues while using rectifiers like ReLU for better functionality.

All of this causes a common choice for image recognition and natural language processing is ReLu activation function. ReLU, for one, if you are developing an image recognition model can be used to find the edges and shapes of objects in an image.

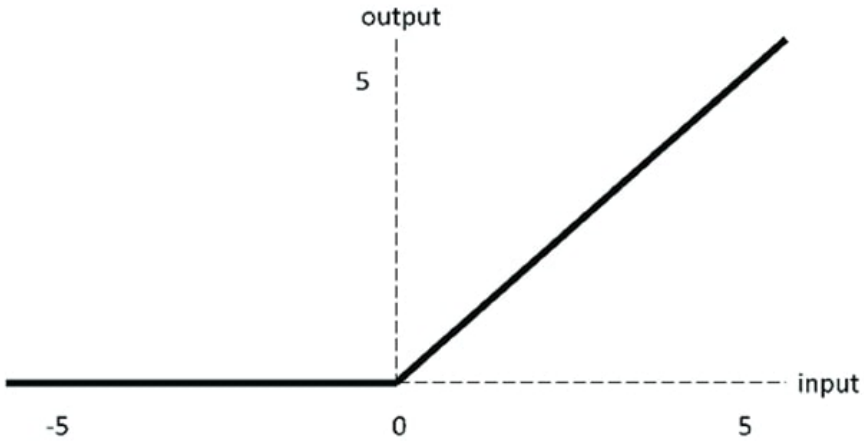

Figure 2.5: ReLu Activation Function

2.3.3. Softmax

The softmax function is an activation function which is used for multi-class classification problems. It transforms the input values into the range from 0 to 1, so that we have a homogeneous space with all probabilities being in such value. The *Equation 2.5* represents the softmax activation function mathematically.

$$f_i(x) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

Equation 2.5: Formula of the Softmax Activation Function

In this case the sum of all probabilities produced by the function will be one. This makes it very suitable for multi-class classification problems. Softmax outputs define probability distribution of each class which helps facilitate interpretability of classifier's outcomes. However, considering computation aspect; compared to sigmoid and ReLU, softmax is more

complicated. In softmax however output values are normalized while this is not the case with sigmoid functions. Therefore, when working with large datasets, computational costs can increase while using softmax.

Soft-max, also known as logistic regression is often used especially in image classification (LeCun, et al., 1998), natural language processing (NLP) (Mikolov, et al., 2013) and speech recognition (Hinton, et al., 2012) due the fact that it has been widely used widely adopted. For instance, using the softmax function could help to determine if an image was either a cat, a dog or even bird when creating an image classification model. The graph of the softmax activation function can be found in *Figure 2.6*.

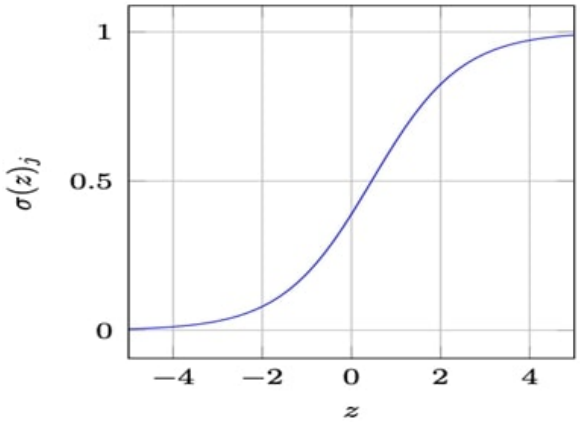

Figure 2.6: Softmax Activation Function

2.4. Classification

In machine learning, classification is a critical and pervasive task, which refers to the division of instances in a dataset into predefined classes or categories. Its applications range from medicine through engineering to finance and marketing. In general, the two major categories of classification issues are binary as well as multi-class classifications. Binary classification is where an example falls under only one of the two classes while multi-class involves many classes.

There have been several algorithms devised in this approach. K-Nearest Neighbors (KNN), Naïve Bayes, decision trees, Support Vector Machines (SVM), and Artificial Neural Networks (ANN) are some of these algorithms. All these algorithms have their performance characteristics, making them applicable to certain types of issues.

The right selection of a classification algorithm depends on features within the dataset as well as the problem itself. These factors are constituted by dataset's size and complexity, number of classes, computational cost, interpretability among others.

In various areas like image classification, natural language processing, speech recognition, medical diagnosis, fraud detection, customer segmentation as well as product recommendations it is possible to employ classification procedures.

Some of the benefits associated with classification are the possibility of dividing data into meaningful categories; this helps in decision making, automation and efficiency or even enhancing knowledge discovery. However, there may be challenges related to classification such as poor data quality, feature selection issues, class imbalance, and model interpretability which should be taken seriously during classification.

Classification remains an active research area in the field of machine learning. New research areas such as deep learning, active learning, and transfer learning applied to classification enable the development and improvement of classification techniques.

3. LITERATURE REVIEW

When reviewing previous studies, it has been found that there are no innovative artificial intelligence-based systems for detecting ochratoxin in dried grapes. Previous studies have relied on chemical analyses and methods. While there are studies on artificial intelligence-based methods for various other products, there is no prior research specifically related to this project.

In a conducted study, the presence of Ochratoxin A (OTA) in grapes and grape products available for sale in Istanbul was determined using the reverse-phase high-performance liquid chromatography (HPLC) method after extraction and purification with immunoaffinity columns. The analyses were conducted isocratically at a flow rate of 1 mL/min using a water:acetonitrile:acetic acid (99:99:2, v:v:v) mixture as the mobile phase, with fluorescence detection at an excitation wavelength of 333 nm and an emission wavelength of 460 nm [7].

In another study, a total of 80 samples of dried grapes were used, with 30 samples obtained from İzmir, 30 from Aydın, and 20 from the province of Manisa in the Aegean Region, all acquired from public sales points in their original packaging. The samples were transported to the laboratory under cold chain conditions and analyzed on the same day. They were kept at 4°C until the analyses were completed. The samples underwent mold analysis followed by ochratoxin A (OTA) analysis under aseptic conditions.

Mold analysis was conducted according to the FDA BAM Chapter 18 method. Samples were weighed (25 g) and transferred to 225 mL of 0.1% peptone solution, then homogenized using a stomacher. A 1 mL inoculum was taken from the mixture and spread onto dried and pre-dried DRBC agar plates using a sterilized L rod. The plates were incubated at 25°C for 5 days, and after incubation, the colony-forming units per milliliter (CFU/mL) were calculated using the dilution factor.

For the investigation of OTA presence, sample extracts were injected into an HPLC device using an acetonitrile:water:acetic acid mixture as the mobile phase. The VICAM method was utilized for OTA analysis, with an Ace 5 C18 column and a Fluorescence EM: 443 nm detector employed for OTA analysis in the samples. The mobile phase flow rate in the HPLC device for OTA analysis was set at 1.2 mL/minute [8]

In another study, high-performance liquid chromatography (HPLC) method was utilized to determine the presence of Ochratoxin A (OTA) and Fumonisin B2 (FB2) in dried grapes. Following the extraction of samples, the extracts purified with an immunoaffinity (IA) column were analyzed using the HPLC method. After the separation of OTA and FB2 in the reverse phase column, detection and quantification were performed using a fluorescence detector. The obtained results were evaluated according to the maximum limits accepted in the EU countries and Turkish Food Codex [9].

Among the analytical methods used for the determination of Ochratoxin A (OTA) in food items, it is known that OTA is a derivative of L-phenylalanine-bound isocoumarinic acid and exhibits optical activity. Due to these characteristics, OTA is determined using chromatographic techniques. Particularly, high-performance liquid chromatography (HPLC) and liquid chromatography with fluorescence detection (LC/FLD) are the most used methods. Alternatively, other analysis methods such as thin-layer chromatography (TLC), gas chromatography-mass spectrometry (GC-MS), capillary electrophoresis combined with laser-induced fluorescence, and enzyme immunoassay (EIA) are also employed [10].

Methods based on fluorescence detection liquid chromatography have been officially recognized as the standard method for OTA determination. These methods are used to measure the quantity of OTA in various food items such as grains, coffee, wine, and beer. Additionally, ELISA kits are commonly used for OTA screening. New food technologies propose various methods that enable the rapid analysis of OTA. LC/MS can be used as an alternative to LC/FLD. However, LC/FLD is considered a more convenient, economical, and sensitive method besides having lower limits of detection (LOD) [11].

4. DESIGN

This section will address the factors influencing the detection of ochratoxin in raisins. The cost of this project will be calculated, and its environmental, ethnic, social, and political impacts will be discussed. The overall structure for detecting ochratoxin in raisins will also be outlined. Firstly, a workflow was devised, and the process proceeded according to this workflow. Subsequently, it will be examined in more detail in the subsections.

4.1. Realistic Constraints and Conditions

When evaluating the constraints and conditions for detecting ochratoxin in raisins using NIR spectrometry in a thesis, various factors need to be considered. There are several practical constraints and limitations that may be applicable in the subsequent subsections.

Economy: The economic analysis of DLP210NIR SCAN Nano device should consider the potential profits and pricing issues associated with the project outputs.

When getting to know the total cost one should first pay attention to how much the gadget costs initially, then how much it's going to take for its maintenance, repairs and regular calibration. The lifespan and quality of parts comprising the device may also impact on total ownership costs.

The DLP210NIR SCAN device can be a great financial contributor in cases of material sorting and quality control. It's rapid enough spectrum analysis to enhance production efficiency, reduce wastages and improve product quality that will result in increased customer satisfaction. Through this, companies may get the ability to give more reliable products on the home market as well as gain the competitive edge.

Furthermore, adoption of this technology may increase market share and promote innovation. For example, accelerated and better precision separation in food agriculture and healthcare applications such as those being developed in this project will contribute towards productivity improvement while reducing pollution which would attract more people to buy the gadget. The same could even open doors for export into foreign markets, hence increasing competitiveness in the internal market.

Cost Analysis: Cost analysis aims to detail the labor, raw materials, required machinery and equipment, procurement of services, and processing costs that arise within the project. A comprehensive cost analysis related to the DLP210NIR SCAN device can assist in evaluating the overall impact and financial viability of the project.

The use of the DLP210NIR SCAN device can lead to increased efficiency by enabling rapid and accurate analysis of materials. This increased efficiency can contribute to reducing labor costs by reducing the time required for manual inspection or analysis. The device's ability to optimize processes can reduce overall costs by reducing the need for additional machinery or equipment. Additionally, the near-infrared (NIR) spectroscopy capabilities of the device can enhance product quality by allowing the detection of contaminants or pollutants. This improved quality control can reduce the need for reprocessing or recalls, thereby saving costs in terms of raw materials and labor.

However, the high initial investment cost of the DLP210NIR SCAN device is a significant disadvantage. The initial expenses, such as purchasing the device and providing training to operators, are high. Additionally, regular maintenance and calibration are required to ensure the accuracy and reliability of the device. These ongoing costs can impact the project budget and overall cost-effectiveness.

Environmental Issues: In this project, NIR spectroscopy will be used for the detection of ochratoxin in raisins. Within the scope of this study, measures should be taken to reduce energy consumption, prevent the release of harmful substances into the environment, and promote energy efficiency and recycling to achieve the project output.

In particular, the energy efficiency of the NIR spectrometer should be considered. Appropriate energy management strategies should be implemented to optimize the device's energy consumption. Recyclable and environmentally friendly options should be preferred in the selection of materials used in the production process.

Measures should be taken to prevent the release of waste products and harmful substances into the environment. Environmentally friendly methods should be preferred to minimize waste generation, and energy consumption should be reduced by using alternative energy sources.

Various measures should be taken to ensure that this project adheres to sustainability principles. These measures should be considered and implemented throughout the project process.

Sustainability: Studies addressing topics such as the durability of selected materials in product design, their potential for waste generation, and recyclability provide crucial insights into sustainability. The DLP210NIR SCAN device stands out as a significant tool contributing to research in this field.

Utilizing near-infrared (NIR) spectroscopy, the device enables rapid analysis of material composition. This analysis plays a vital role in assessing the sustainability of materials used in product design. By identifying recyclable components and detecting harmful substances, it facilitates the development of eco-friendly and recyclable products. Moreover, it enables the swift and accurate separation of different types of materials when used in sorting and recycling facilities. This paves the way for efficient recycling processes and supports sustainable waste management practices.

However, the high initial cost of the device is a notable disadvantage. The expense associated with the necessary equipment and training for proper operation may pose a barrier for small-scale businesses. This limitation could restrict the widespread adoption of the technology in certain industries.

Therefore, the use of the DLP210NIR SCAN device in sustainability-focused projects can significantly contribute to the development of environmentally friendly products and the achievement of sustainability goals.

Manufacturability: In the project for detecting ochratoxin in raisins, it is essential to consider manufacturability, a significant factor determining the suitability of a design for the production process, in conjunction with the DLP210NIRSCAN Nano device.

Firstly, appropriate material selection is crucial. Materials compatible with the device's intended use should be chosen, ensuring they do not interfere with the analysis process. Subsequently, the production method should be determined. A production method compatible with the selected materials and the device's design should be chosen, considering cost and efficiency.

Production stages should also be carefully planned. Steps such as manufacturing, assembly, testing, and calibration of components should be identified and carried out in the appropriate sequence. This ensures the production process progresses smoothly and effectively.

Adhering to manufacturability principles during the project ensures the efficient production of the device. By selecting the right materials and employing suitable production methods, project costs can be reduced, and time wastage can be prevented. Additionally, this approach facilitates the successful completion of the project.

Ethical: Ethical: The use of NIR spectroscopy for ochratoxin detection in raisins does not pose ethical issues. However, when employing different imaging techniques, such as aerial imaging, for spectrum scanning in public areas, utmost care should be taken to avoid any violation of personal data protection laws.

Health: Ochratoxin is known to have harmful, toxic, and carcinogenic effects on human health. Therefore, it is crucial for the project manager to take necessary precautions and ensure sterilization during close-contact data collection studies conducted on both healthy and ochratoxin-contaminated raisins. Raisins should be placed in transparent sealed bags, and the use of gloves and masks should be emphasized during the study.

Safety:

1**. Engineering Safety for NIR Spectrometer Usage**:

- Proper installation and operation of the NIR spectrometer are essential, ensuring secure electrical connections during setup.

- Additionally, users should be present during spectrometer operation to monitor operations and promptly detect and prevent measurement errors.

2. **Occupational Health for NIR Spectrometer Usage**:

- Operators must use personal protective equipment (e.g., goggles, gloves, masks) during spectrometer usage to mitigate chemical or physical hazards.

- Adequate training and instructions should be provided for safe and effective spectrometer usage.

3. **Engineering Safety for Raspberry Pi Usage**:

- Attention should be given to electrical safety measures during Raspberry Pi usage, ensuring safe connection and operation.

- Proper cooling and ventilation of the Raspberry Pi device are necessary to prevent overheating and associated risks.

4. **Occupational Health for Raspberry Pi Usage**:

- Adherence to workplace safety procedures is essential during Raspberry Pi usage. Operators should be trained to recognize hazards and take appropriate precautions.

Social and Political: The use of technologies such as NIR spectroscopy and Raspberry Pi for ochratoxin detection in raisins should align with social and political values. These projects reflect significant societal values related to food safety and public health. Additionally, the adoption of such technologies can contribute to social development by enhancing agricultural sector efficiency. Politically, support for such projects should align with the country's food safety policies and support strategic objectives aimed at improving overall societal welfare. Therefore, the use of technologies like NIR spectroscopy and Raspberry Pi should be conducted in harmony with social and political values, focusing on societal needs and interests.

4.2. Cost of the Design

In this project, support has been obtained from our school and companies in the industry for the procurement of necessary equipment. The potential costs of the basic equipment used will be discussed in the following section.

DLP NIR SCAN NANO: $1450

Raspberry Pi 4: ~$150

Google Colab Pro: $9.99 Per month

DLP NIR Scan Nano Black Box Design and consumables: ~$60

This potential cost analysis has been designed to cover the minimum costs required for the development of the project.

4.3. Engineering Standards

1. **Raspberry Pi for image processing education (10.23919/EUSIPCO.2017.8081633):**
This standard addresses the usage of Raspberry Pi for image processing training, providing guidelines and best practices for the use of Raspberry Pi platforms in an educational context. This standard has been referenced in the design of the DLP210NIR device to ensure that its image processing capabilities are aligned with educational objectives and tailored for academic environments. Within the scope of the project, the low-cost, portable, and user-friendly nature of Raspberry Pi has been aimed to be utilized as an effective tool in image processing training.

2. **Data classification with deep learning using TensorFlow (10.1109/UBMK.2017.8093521):**
This standard addresses data classification using deep learning with TensorFlow, providing insights into the application of deep learning algorithms for data classification tasks. Within the project context, this standard has guided the integration of TensorFlow-based deep learning models for image classification and ensured the accurate classification of data obtained through NIR spectroscopy by the device. This has played a significant role in enhancing the device's ability to detect harmful substances such as mycotoxins in materials, thereby contributing to achieving reliable results.

3. **Programming language Python for data processing (10.1109/ICECENG.2011.6057428):**
This standard addresses the use of the Python programming language for data processing, outlining best practices for coding in Python to provide efficient and effective data processing workflows. Within the context of our project, this standard has been considered to ensure that the implemented data processing algorithms are well-structured, modular, and optimized for performance. Within the project scope, Python's flexible and powerful nature has been acknowledged as an ideal language for effectively carrying out data processing tasks, and it has been utilized in the development of the artificial intelligence-based model used for detecting harmful substances such as mycotoxins.

4.4. Details of the Design

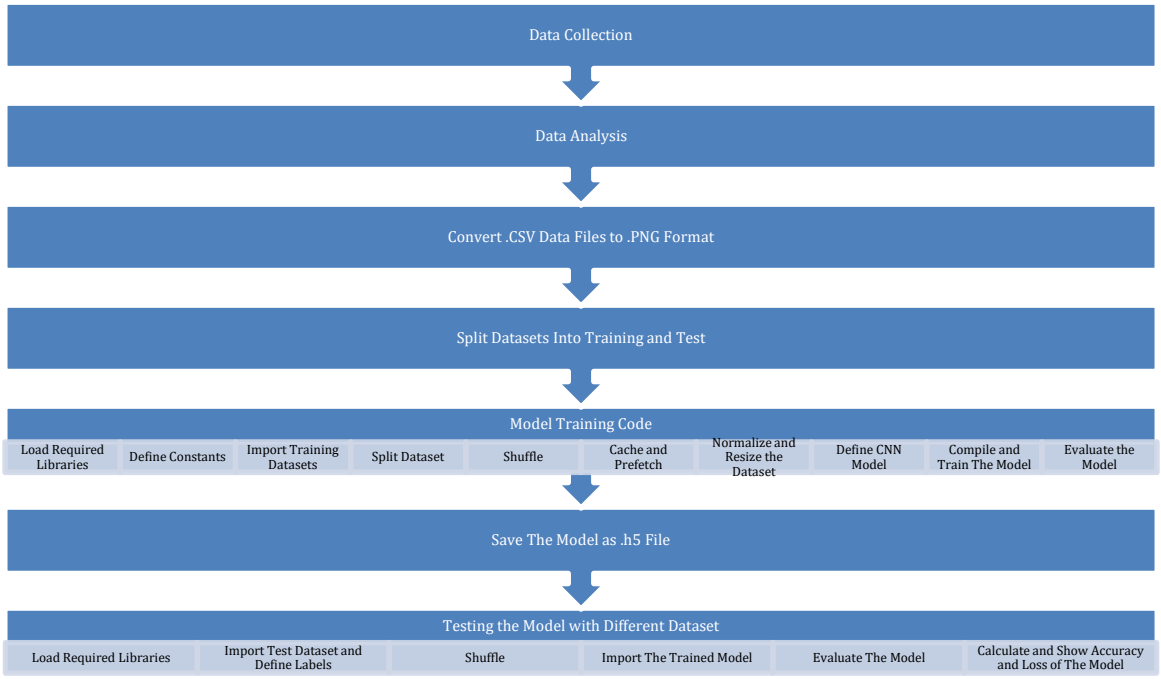

Figure 4.1: Flowchart of The Project

The processes carried out during this project are shown in *Figure 4.1* as a flowchart. All steps are explained in detail below.

DATA COLLECTION:

In this study, data collection was conducted using the DLP NIR Scan Nano spectrometer. In each data collection step, a total of three different .CSV files were obtained, including Absorbance, Reflectance, and Density. However, this project focused only on Absorbance and Reflectance data.

The data obtained by the spectrometer is crucial for determining the amount of ochratoxin present in dried grapes. Absorbance and Reflectance data reflect the spectral characteristics of dried grapes, and the examination of these characteristics is an important criterion for detecting the presence of ochratoxin.

The preference for Absorbance and Reflectance data in the project stems from the fact that these data provide important information about the molecular structures of dried grapes. The utilization of this data has contributed to enhancing the accuracy and reliability of artificial intelligence-based systems in ochratoxin detection.

The Absorbance and Reflectance data obtained during the data collection process were used in training the artificial intelligence model developed within the project. These data served as the primary source of data for the model to accurately detect the presence of ochratoxin. Therefore, the quality and accuracy of the Absorbance and Reflectance data obtained during the data collection phase have a decisive impact on the success of the project.

DATA ANALYSIS:

In this section, the entire dataset containing Absorbance, Reflectance and Wavelength data in CSV files was analyzed and erroneous data was deleted. Also, it was decided whether the data set was decomposable or not, before starting the training. At the same time, the data was plotted collectively on a single graph and the data distribution, and the presence of mismatched data were checked. The mismatched data or erroneous data were removed from the data set and the training process started with a healthy and error-free data set.

CONVERT .CSV DATA FILES TO .PNG FORMAT:

The data files obtained in .CSV format from the data collection process were processed and converted to .PNG format at this stage. Initially, the collected data was organized into a separate folder, which was then imported for use in Python code. Subsequently, using the Python programming language, these data were visualized on a single graph.

The generated graph has an axis layout featuring wavelength (X-axis) against Absorbance (Blue) and Reflectance (Red) values. While the X-axis represents the wavelength of light, the Y-axis includes units (AU) denoting Absorbance and Reflectance values. The Blue color on the graph represents Absorbance, while the red color represents Reflectance values.

This transformation process facilitated the visual analysis of the obtained data and allowed for a more comprehensible examination of the dataset. Additionally, the creation of graphs prepared the dataset for use in training artificial intelligence models.

SPLIT DATASET INTO TRAINING AND TEST:

At this stage, the generated graph images in .PNG format were divided into training and test datasets. There are a total of 700 graph images, which have been divided into two separate folders: 500 images for training and 200 images for testing.

The separation of training and test datasets determines the data to be used for training and validating the artificial intelligence model. The training dataset is used to learn and adjust the parameters of the model, while the test dataset is used to evaluate the performance of the trained model on previously unseen data.

TRAINING CODE PREPARATION:

Here are the step-by-step details of the process for training the Convolutional Neural Network (CNN) model:

1. Downloading Required Libraries: Python libraries necessary for the proper functioning of the code were downloaded. These libraries contain functions required for creating the model, data processing operations, and the training process.

2. Importing the Dataset: A total of 1000 data, consisting of 500 images of ochratoxin-infected and 500 images of healthy dried grapes previously designated for training, were

imported into the Python environment. This dataset forms the primary data source for training the model.

3. Defining Constants: Constants necessary for the code to function properly were defined. These constants include values such as image dimensions, the number of training epochs, and early stopping patience.

4. Splitting and Preprocessing the Dataset: The imported dataset was split into training, validation, and test sets in different proportions. In this step, the pixel values of the images in the dataset were normalized and rescaled. Additionally, the dataset was cached and shuffled. These steps are important to ensure a more stable training process.

5. Creating the CNN Model: A CNN model was created for training. The model was designed to work with image dimensions of 512x512.

6. Training and Model Improvement: The training process was initiated with the number of epochs initially set to 300. The Early Stopping method was applied to stop training when no improvement in the loss value of the model was observed. In this method, the patience level was set to 10. Thus, training was automatically stopped when there was no improvement in the loss value of the model, and the best model parameters were saved.

7. Testing and Saving the Model: The trained model was evaluated with the test dataset, and the performance of the model was assessed. Subsequently, the model demonstrating the best performance was saved.

TESTING THE MODEL WITH DIFFERENT DATASET:

Here are the steps followed to test the previously trained model with a different dataset:

1. Downloading Required Libraries: Initially, the necessary Python libraries for conducting the testing process were downloaded. These libraries contain functions to be used during the testing process.

2. Importing the Dataset and Determining Labels: The datasets of ochratoxin-infected and healthy dried grapes, previously separated into folders, were imported into the Python environment. The labels of the images in these datasets were determined and classified.

3. Shuffling the Datasets: The imported datasets were shuffled to ensure a more reliable and objective testing process. This ensured that the model would evaluate its performance more generally.

4. Importing the Trained Model: The previously trained model file was imported into the Python environment. This step is necessary for the model to make predictions on the test dataset.

5. Testing the Model and Performance Evaluation: The trained model was applied to the imported test datasets. The predictions of the model were compared with the actual label

values to calculate accuracy rate and loss values. These values were used to objectively evaluate the performance of the model on the test dataset.

With the completion of these steps, the ability of the trained model to generalize and its performance on different datasets were objectively evaluated. These evaluation results provide valuable insights into how effective the model could be in real-world applications.

5. METHODS

In this study, spectral analysis of ochratoxin-containing and healthy raisins and classification using machine learning methods were performed. The project steps are described in detail below:

5.1. Sample Preparation

In the first step, ochratoxin-contaminated and healthy raisins were individually packed in sealed transparent plastic bags in a sterile environment *Figure 5.1*. In this procedure, hygiene rules have been strictly adhered to avert contamination and ensure accurate data. In this context, masks were worn, and surgical gloves were used, paying utmost attention to the risk of contact and contamination.

Figure 5.1: Raisin Samples

5.2. Spectral Data Collection

Spectral data of the packaged raisins were collected with a DLP NIR SCAN NANO spectrometer in a black box with a wavelength range of 900-1700 nm, 40 number of scans and 228 digital resolution settings. *Figure 5.2* shows the settings.

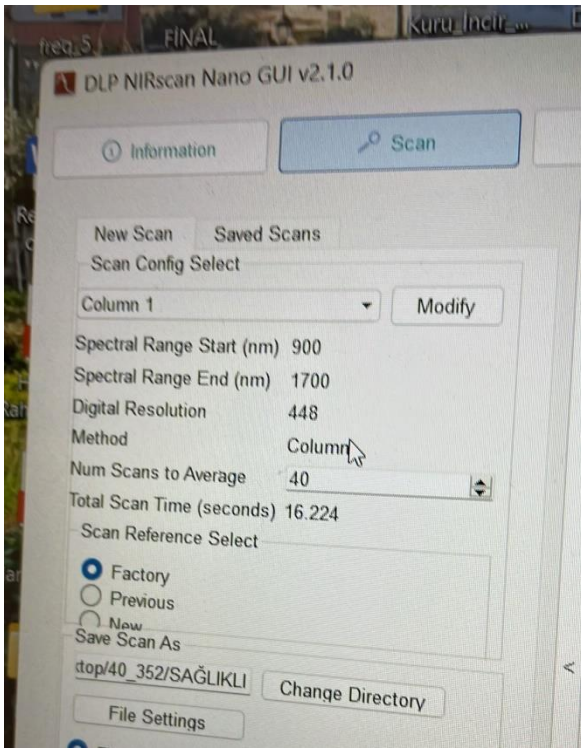

Figure 5.2: NIR Spectrometer Settings

The NIR spectrometer was placed in contact with the raisins with a plastic bag between the raisins and two back-to-front data were obtained from each raisin. In total, 700 data were collected from 350 raisins for each class. Thus, 1400 data were obtained from 700 raisin for two classes in total (*Figure 5.3*).

Figure 5.3: Data Collection in Blackbox

In this step, different data collection environments were tested to find the most suitable environment for analysis with the NIR spectrometer. There are two stages in these experiments: white background in *Figure 5.4* and closed black box in *Figure 5.5*. For the black closed box in *Figure 5.5*, the inside of the shoebox was first painted with a white marker.

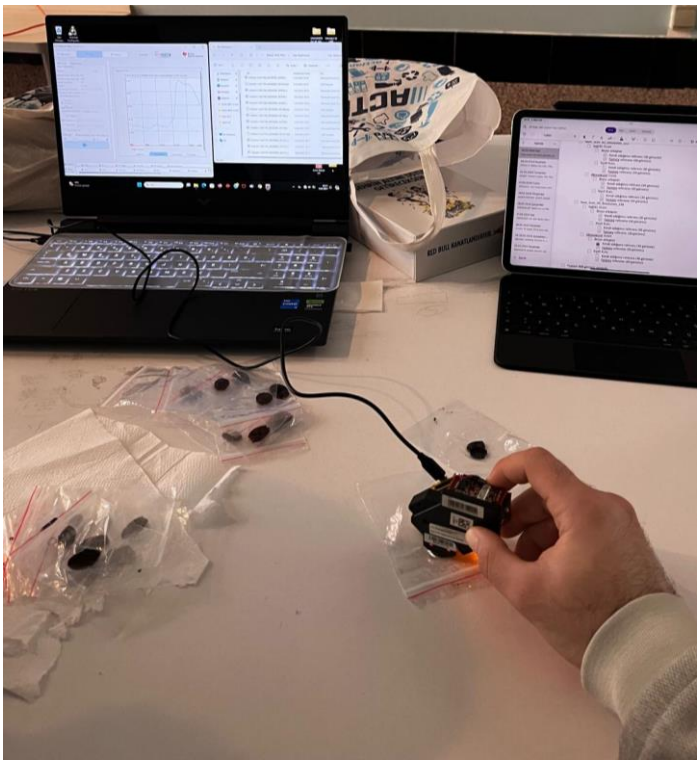

Figure 5.4: Data Collection on White Background

Figure 5.5: Primitive Black Closed Box

When it was determined that there were healthier results in the black closed box, a special box design, shown in *Figure 5.6*, was provided to collect data in a more stable environment by preventing the entry of light into the environment. Thus, the data set was obtained in this black enclosed box by preventing the entry of light from outside and obtaining a healthy data set.
For the black sealed box, the raisins are first pressed on the sensor shown in *Figure 5.7* by placing the raisins in a plastic bag and then the lid of the box is closed, and the data is collected by measuring in a light-free environment.

Figure 5.6: Black Box

Figure 5.7: NIR Spectrometer Sensor

5.3. Data Analysis and Visualization

The collected spectral data was saved in CSV format. Data analysis was applied by using Python so that it could make sure that this information is accurate and consistent. Data analysis included max, min, std dev, absolute value, 25%, 50%, 75% and 100% values (*Figure 5.8*, *Figure 5.9*). According to the information obtained from the analyses, the average absorbance value of the Ochratoxin contaminated raisins was 0.10AU lower and the average reflectance value was 0.018AU higher, indicating an overall difference for the two different classes.

Statistical Summary of OTA Dataset:

	Wavelength (nm)	Absorbance (AU)
count	159600.000000	159600.000000
mean	1321.229994	1.127609
std	232.038478	0.271450
min	901.663021	0.484387
25%	1123.785256	0.910440
50%	1331.407989	1.070072
75%	1523.877216	1.355760
max	1700.708699	1.984677

Statistical Summary of Healthy Dataset:

	Wavelength (nm)	Absorbance (AU)
count	159600.000000	159600.000000
mean	1321.229994	1.229262
std	232.038478	0.277569
min	901.663021	0.550670
25%	1123.785256	1.008351
50%	1331.407989	1.159220
75%	1523.877216	1.457821
max	1700.708699	2.209605

Figure 5.8: Absorbance Statistical Summary of The Dataset

Statistical Summary of OTA Dataset:

	Wavelength (nm)	Reflectance (AU)
count	159600.000000	159600.000000
mean	1321.229994	0.088923
std	232.038478	0.049616
min	901.663021	0.010359
25%	1123.785256	0.044080
50%	1331.407989	0.085100
75%	1523.877216	0.122902
max	1700.708699	0.327803

Statistical Summary of Healthy Dataset:

	Wavelength (nm)	Reflectance (AU)
count	159600.000000	159600.000000
mean	1321.229994	0.070666
std	232.038478	0.039264
min	901.663021	0.006172
25%	1123.785256	0.034848
50%	1331.407989	0.069307
75%	1523.877216	0.098095
max	1700.708699	0.281404

Figure 5.9: Reflectance Statistical Summary of The Dataset

After the analyses, absorbance and reflectance data were plotted on a single graph to examine the consistency and distribution of the data. After these analyses, missing and erroneous data were identified and removed, and reliable data were collected instead. The graphs obtained at this stage are shown below in *Figure 5.10*, *Figure 5.11*, *Figure 5.12*, *Figure 5.13*. The mismatched data seen in the healthy grape data (*Figure 5.12*, *Figure 5.13*) were removed. As can be seen from the aggregated graphs plotted on a single graph, there is a significant change in Absorbance and Reflectance values in the wavelength range 1400nm to ~1670nm.

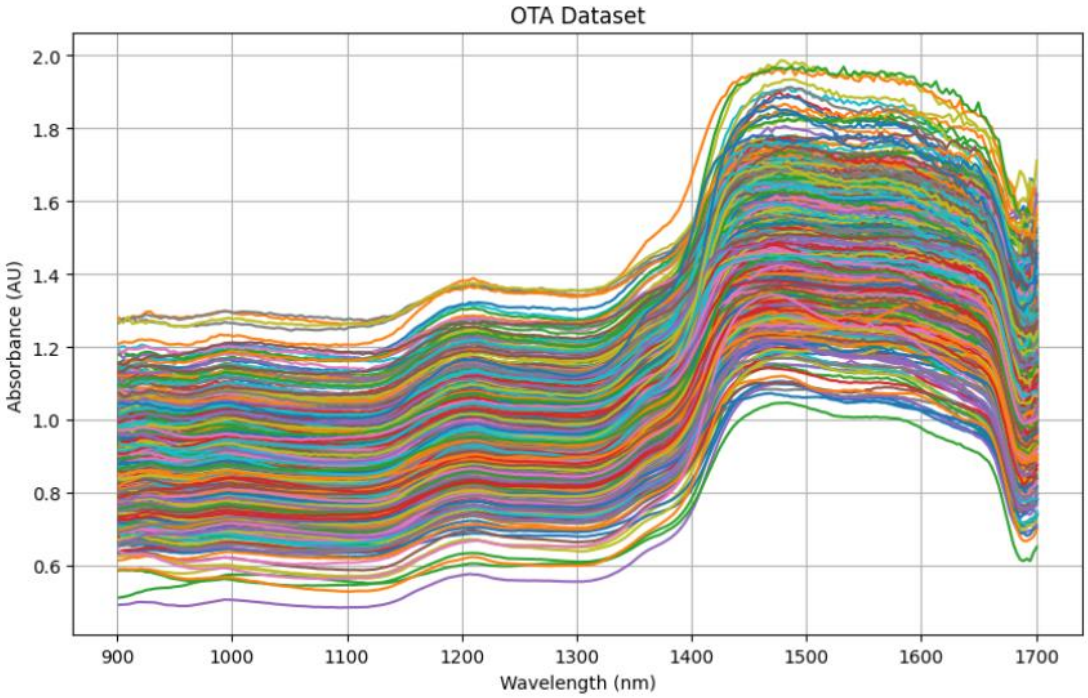

Figure 5.10: OTA Dataset Absorbance Distribution Graph

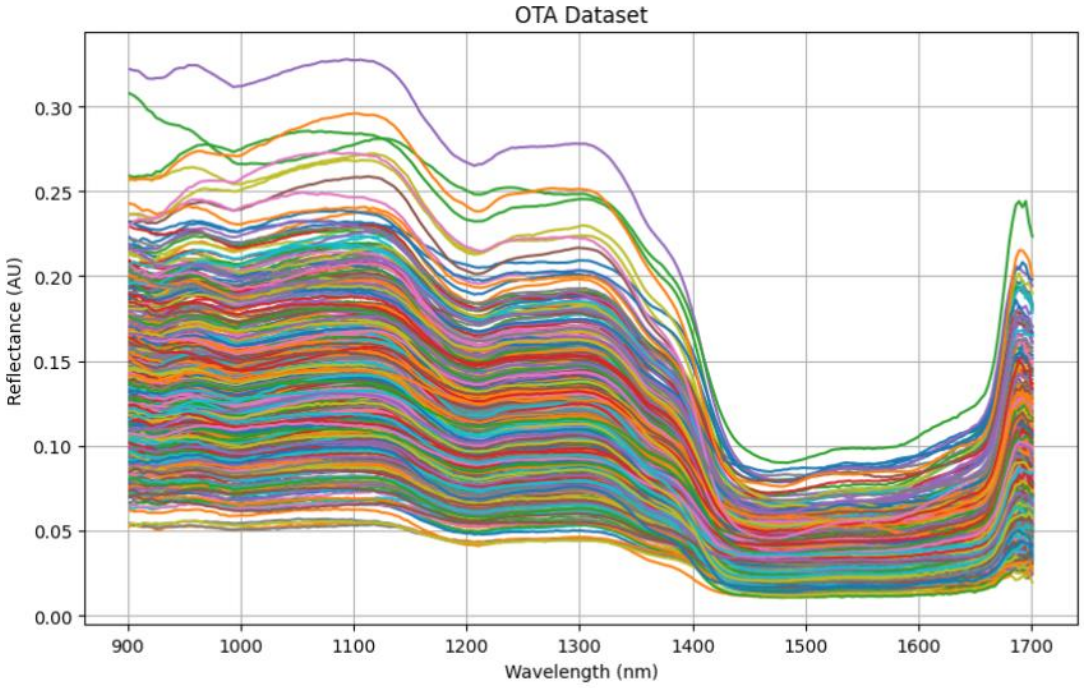

Figure 5.11: OTA Dataset Reflectance Distribution Graph

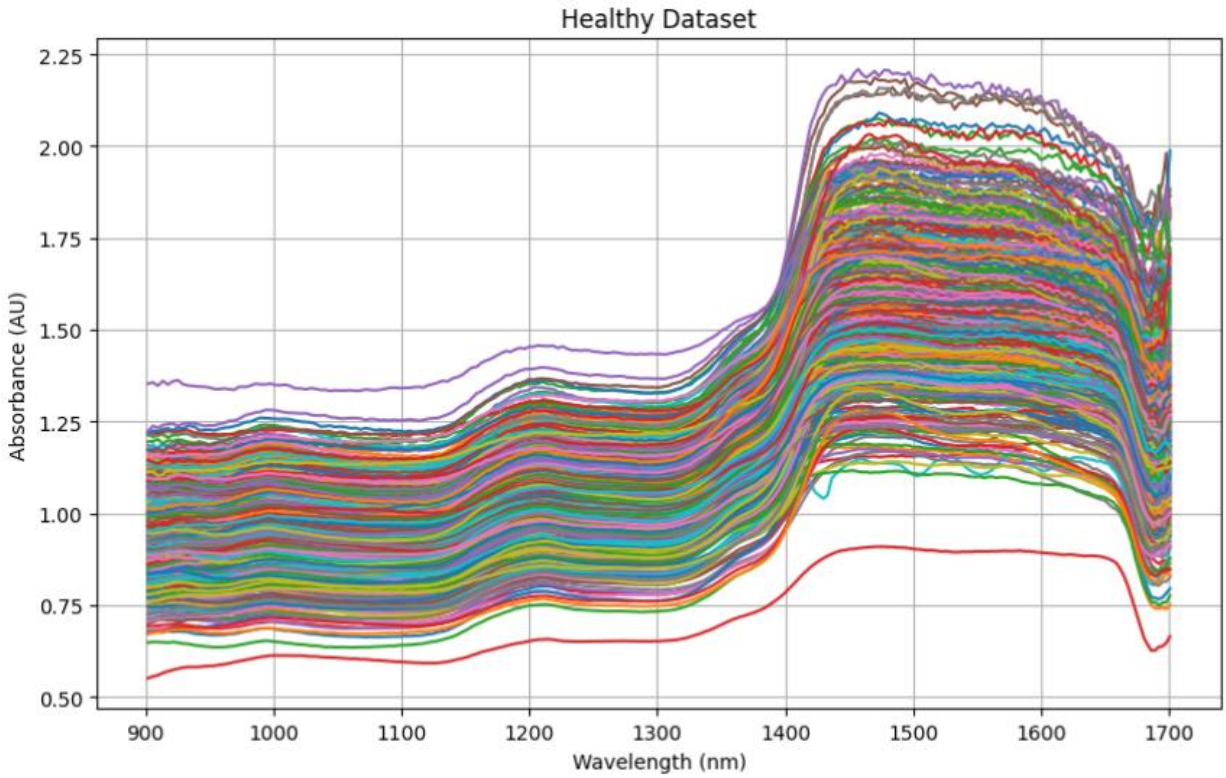

Figure 5.12: Healthy Dataset Absorbance Distribution Graph

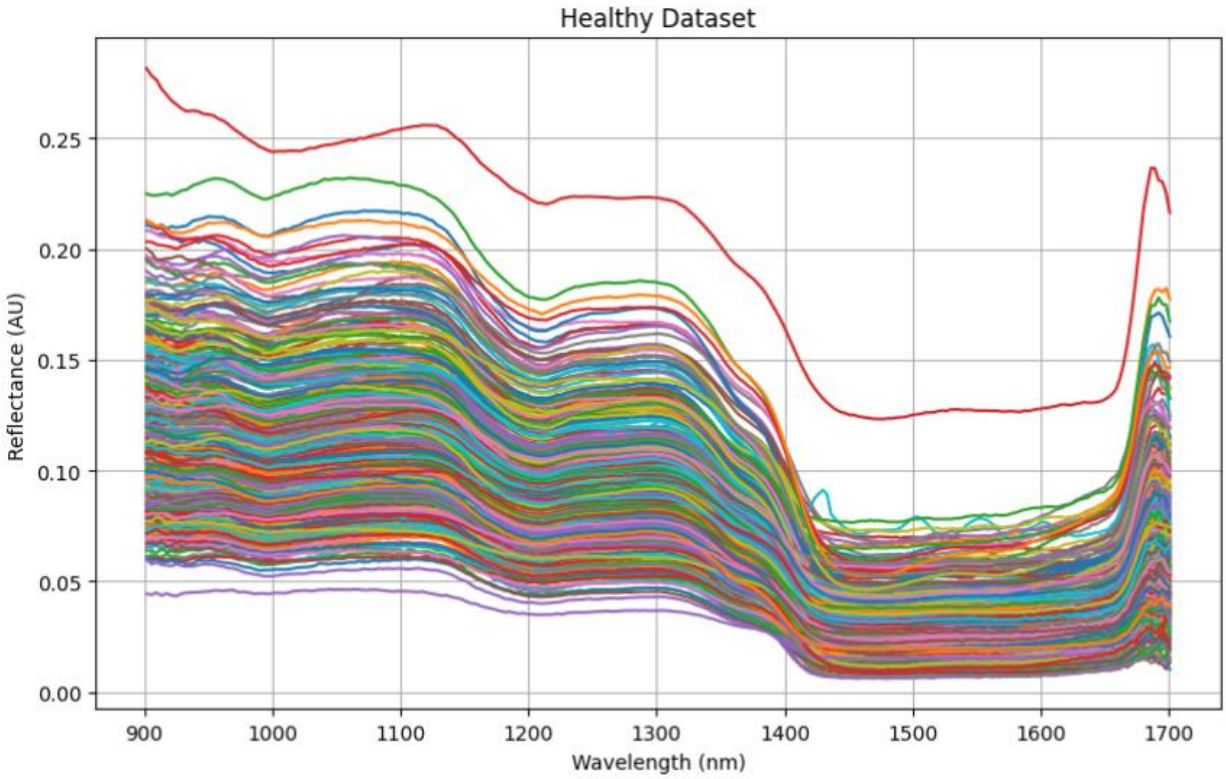

Figure 5.13: Healthy Dataset Reflectance Distribution Graph

After the analysis, the .CSV files that make up the data set were converted into .PNG image files with 512x512 pixel dimensions on a single graph containing Absorbance and Reflectance data for each measurement shown in *Figure 5.14* and *Figure 5.15*. Thus, the change of two different data (Absorbance and Reflectance) in a single image could be observed. Two different experiments were performed in this step. These included plots with no limitation on the axes shown in *Figure 5.14*, and plots with a fixed limitation in the range of 0-2.5AU for the Y axis and 900-1700nm for the X axis shown in *Figure 5.15*.

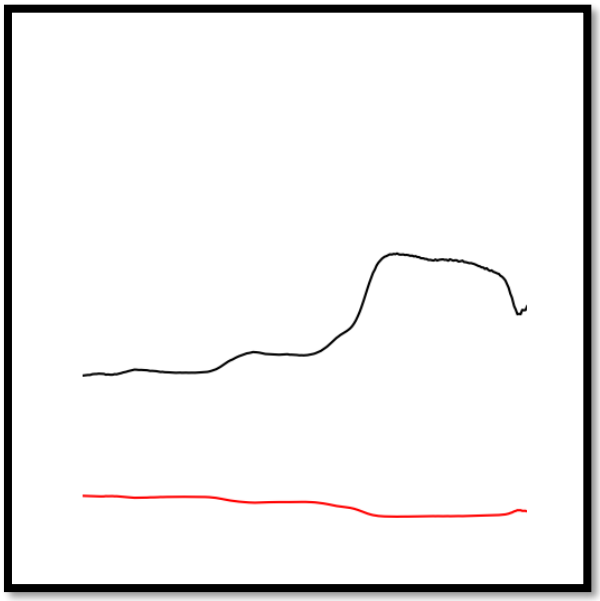

Figure 5.14: Healthy Limited Graph

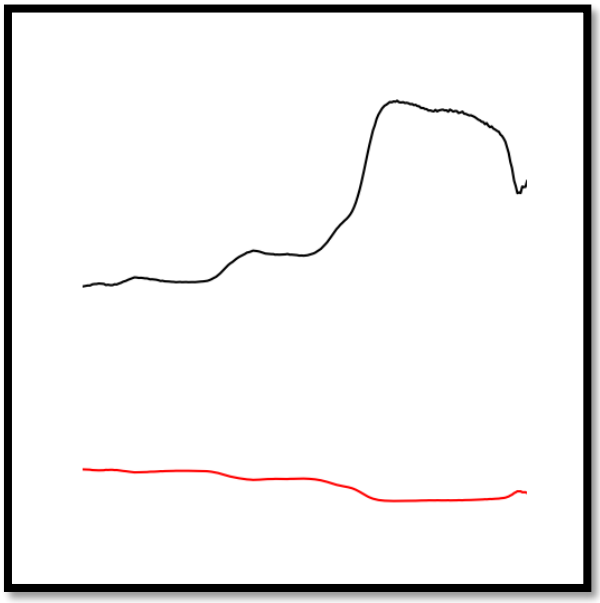

Figure 5.15: Healthy Unlimited Graph

5.4. Model Training and Testing Process

The prepared images were divided into 500 training and 200 test data sets. The training data set was divided into 80% training, 10% validation and 10% test sets. In this part, 70% training, 20% validation and 10% test sets were also tried, but the best result was obtained in the condition divided as 70% training and used to train the selected models. Thus, the dataset was used effectively. Python programming language, Visual Studio and Google Colab platform were used in the training process of the models. Coding details are discussed in detail under the title "Design of the Details".

Optimal hyperparameter settings were implemented in training the models to come up with models of high performance. Various machine learning classification algorithms like, VGG16, ResNet50, SVM, KNN, EfficientNet, XGBoost, LSTM were used during the experiments. In terms of accuracy, VGG16 and LSTM was the best performing among all the classification models For the VGG16 model, experiments were performed with data augmentation, blurriness, single absorbance graph and single graphs of absorbance and reflectance, but the accuracy rate obtained did not exceed 80-90% for training accuracy in the most optimal case. Nevertheless, the new customized CNN model produced the best output. In this model, 99% training accuracy and 17% loss were observed. When the saved model was tested with a new dataset containing 200 data not seen by the model, 85% test accuracy was achieved.

Now it will be described the structure of a convolutional neural network (CNN), the layers of the model, the hyperparameters and the preprocessing steps in detail.

5.4.1. Preprocess:

Preprocessing steps were applied to fit the model's input data. Thus, the data is used more efficiently in the training process. In this section, the data set was loaded, classified, split normalized and several different Data Augmentation techniques were used to obtain the optimal result (*Figure 5.16*).

BATCH_SIZE, IMAGE_SIZE, CHANNELS, EPOCHS: These constants are the training parameters of the model. The size of data to be processed in each step is determined by the batch size while image size and channels, which determine the size of the input images and channel count.

tf.keras.preprocessing.image_dataset_from_directory: loads and classifies images as TensorFlow dataset objects.

seed: Positions the shuffling order of datas within the dataset. Thus, if the function is called again using the same seed value, the same mixing order is obtained each time. This ensures that model training is repeatable. This means that when you train your model on different computers or at different times, you get consistent results every time.
shuffle=True Allows random shuffling of the data set.

Using seed and shuffle Parameters Together: The data set can be shuffled using the same seed value in each epoch. This ensures that the model encounters a different data sequence in each epoch, but follows the same general shuffle sequence each time.

```
BATCH_SIZE = 64
IMAGE_SIZE = 512
CHANNELS=3
EPOCHS=400



"""### Import data into tensorflow dataset object"""

dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "/content/drive/MyDrive/AliBaki_TURKOZ_Engineering_Project/Data_Sets/Grafikler/Çift_Grafikler/700_Sinirsiz_Cift/500_Train",
    seed=12,
    shuffle=True,
    image_size=(IMAGE_SIZE,IMAGE_SIZE),
    batch_size=BATCH_SIZE
)

class_names = dataset.class_names
class_names
```

Figure 5.16: Preprocessing and Constants

5.4.2. Data Visualization

By visualizing some images in the dataset, it is checked that the dataset is loaded correctly, and the classes are correctly separated. This avoids wasting time with incorrect training processes. Here, a bit batch is taken from the dataset and the images and labels in this batch are visualized in 20x20 dimensions by iteration (*Figure 5.17*).

```
"""### Visualize some of the images from our dataset"""

plt.figure(figsize=(20, 20))
for image_batch, labels_batch in dataset.take(1):
    for i in range(12):
        ax = plt.subplot(3, 4, i + 1)
        plt.imshow(image_batch[i].numpy().astype("uint8"))
        plt.title(class_names[labels_batch[i]])
        plt.axis("off")
```

Figure 5.17: Data Visualization

5.4.3. Splitting the Data Set:

The data set is randomly shuffled and divided into 70% training, 20% validation and 10% test sets. Here, ds.skip and ds.take functions are used together to ensure that the train, validation and test data sets are the same every time, whether the shuffle parameter is True or False, and that the experiments are performed in a more controlled environment. The shuffle_size parameter specifies the number of data samples used for shuffling. If shuffle is True, this number of samples is randomly selected and sorted (*Figure 5.18*).

```python
def get_dataset_partitions_tf(ds, train_split=0.7, val_split=0.2, test_split=0.1, shuffle=True, shuffle_size=1000):
    assert (train_split + test_split + val_split) == 1

    ds_size = len(ds)

    if shuffle:
        ds = ds.shuffle(shuffle_size, seed=12)

    train_size = int(train_split * ds_size)
    val_size = int(val_split * ds_size)

    train_ds = ds.take(train_size)
    val_ds = ds.skip(train_size).take(val_size)
    test_ds = ds.skip(train_size).skip(val_size)

    return train_ds, val_ds, test_ds

train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset)
```

Figure 5.18: Train, Test, Validation Data Split

5.4.4. Data Set Caching and Prefetching:

We can improve the performance of the training process by caching and prefetching the dataset. This ensures faster completion of the training process. Cache caches the data set and allows the data to load faster on subsequent uses. Shuffle randomly shuffles the data set by a specified amount, so that the model is equally exposed to the samples in the data set, preventing overfitting. The Prefetch method preloads and buffers the data sets. This allows the model to access the data sets faster, reducing training time. However, Cache and Prefetch methods may increase memory usage and increase training time when working with large data sets. The Tf.data.AUTOTUNE function automatically adjusts the prefetch_buffer_size parameter in the TensorFlow data pipeline by analyzing the hardware and software characteristics of the system. This is designed to ensure optimal performance of the data pipeline (Figure 5.19).

```python
train_ds = train_ds.cache().shuffle(200).prefetch(buffer_size=tf.data.AUTOTUNE)
val_ds = val_ds.cache().shuffle(100).prefetch(buffer_size=tf.data.AUTOTUNE)
test_ds = test_ds.cache().shuffle(50).prefetch(buffer_size=tf.data.AUTOTUNE)
```

Figure 5.19: Cache, Shuffle and Prefetch the Dataset

5.4.5. Resizing, Rescaling and Data Augmentation:

The data has a 512x512 size, and in the image's histogram each pixel value is normalized by dividing it with 255 so that intensity values of every pixel lie within the range 0 to 1. This step ensured that the data was consistent and comparable. Different Data Augmentation techniques were used during the experiments (Figure 5.20).

```
resize_and_rescale = tf.keras.Sequential([
 layers.Resizing(IMAGE_SIZE, IMAGE_SIZE),
  layers.Rescaling(1./255),
])

"""### Data Augmentation
"""

data_augmentation = tf.keras.Sequential([
 #   layers.RandomFlip("horizontal_and_vertical"),
  #  layers.RandomRotation(0.2),
   # layers.RandomZoom(0.2),
     layers.RandomBrightness(0.2),
     #layers.GaussianNoise(0.1),
     #layers.RandomContrast(0.2),
])

"""#### Applying Data Augmentation to Train Dataset"""

train_ds = train_ds.map(
    lambda x, y: (data_augmentation(x, training=True), y)
).prefetch(buffer_size=tf.data.AUTOTUNE)
```

Figure 5.20: Resizing, Rescaling and Data Augmentation

5.4.6. CNN Model:

Conv2D, this layer gives the name of convolutional neural networks. Filters are used for extracting feature maps of images. Each filter is used for extracting specific features in the image. When the number of filters increases, deeper features can be learned. Filters are used to reveal edges, corners, textures, and patterns by calculating the gradient on the image. To apply filters, kernel size, padding and stride values need to be determined. These determine how the filters will be applied. Since padding is not applied in this step in our project, the size decreases by 2 pixels after each convolution layer. For example, in the Figure *5.21*, 5 convolution layers are applied, with the number of filters varying between 32 and 512. A 3x3 filter and ReLu activation function (Equation *2.4*) are applied in each convolution layer. ReLU activation introduces nonlinearity by keeping positive input values intact and setting negative values to zero, thus eliminating the vanishing gradient problem. Equation *5.1* shows the mathematical formula of the Convolutional layer.

$$S_{i,j} = (I * K)_{i,j} = \sum_{m} \sum_{n} I_{i,j} K_{i-m,j-n}$$

Equation 5.1: Formula of Convolutional Layer

MaxPooling2D, this layer reduces the size and computational cost of feature maps. For max pooling, the filter size and stride are determined, and this filter is shifted over the image and the largest of the values in the filter is saved as a single output, thus reducing the feature map size. For example, in Figure *5.21* 2x2 filters are used and the feature map size is halved after each MaxPooling layer.

Flatten: This layer flattens a feature map into a vector, which is needed to connect features from convolutional layers.

Dense: This layer is a fully connected hidden layer. It transforms the incoming feature vector into an output vector of a given size. In the project, ReLU activation (Equation *2.4*) is used in this layer to add non-linear features. But at the last Dense layer Sigmoid activation function (Equation *2.3*) is used for classification. Sigmoid function returns a value between 0 and 1. L2_regularizer is applied as 0.001. The regularization strength increases when its value increases. Networks are made less prone to overfitting in most cases by regularization.

Dropout: This layer reduces overfitting. It updates the weights by randomly dropping (setting to zero) data points at a set rate.

Layer (type)	Output Shape	Param #
sequential (Sequential)	(32, 512, 512, 3)	0
conv2d (Conv2D)	(32, 510, 510, 32)	896
max_pooling2d (MaxPooling2D)	(32, 255, 255, 32)	0
conv2d_1 (Conv2D)	(32, 253, 253, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(32, 126, 126, 64)	0
conv2d_2 (Conv2D)	(32, 124, 124, 128)	73856
max_pooling2d_2 (MaxPooling2D)	(32, 62, 62, 128)	0
dropout (Dropout)	(32, 62, 62, 128)	0
conv2d_3 (Conv2D)	(32, 60, 60, 256)	295168
max_pooling2d_3 (MaxPooling2D)	(32, 30, 30, 256)	0
conv2d_4 (Conv2D)	(32, 28, 28, 512)	1180160
max_pooling2d_4 (MaxPooling2D)	(32, 14, 14, 512)	0
flatten (Flatten)	(32, 100352)	0
dense (Dense)	(32, 512)	51380736
dropout_1 (Dropout)	(32, 512)	0
dense_1 (Dense)	(32, 2)	1026

Total params: 52950338 (201.99 MB)
Trainable params: 52950338 (201.99 MB)
Non-trainable params: 0 (0.00 Byte)

```
input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = 2

model = models.Sequential([
    resize_and_rescale,
    layers.Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(256, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(512, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(512, activation='relu', kernel_regularizer=regularizers.l2(0.001)),
    layers.Dropout(0.5),  # Dropout for regularization
    layers.Dense(n_classes, activation='sigmoid'),
])

model.build(input_shape=input_shape)
```

Figure 5.21: CNN Model

5.4.7. Compiling The CNN Model:

As seen in Figure *5.22*, Adam optimization algorithm and Sparse Categorical Crossentropy loss function were used during the compilation of the model.

Adam Optimizer updates the model parameters using the gradient descent method. Gradient descent uses a technique called momentum to minimize the value of the model's loss function. Momentum speeds up parameter updates by determining the direction and magnitude of updates relative to previous gradients.

Learning Rate: Experiments were conducted with different learning rates up to 0.001, 0.0001, 0.00003. The learning rate determines how much the optimization algorithm changes the weights at each step. Too high a learning rate can prevent the model from reaching the optimal solution, while too low a learning rate can slow down the training process.

Sparse Categorical Crossentropy is used as Loss Function. This function is often used in multi-class classification tasks. To calculate the loss value, it expects the predicted output and the true class to be a single integer. Thus, it calculates the entropy value using the difference between the predicted class and the true class. The calculated entropy value is used as the loss value of the model.

```
# Compile the model with a lower learning rate
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.00001),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)
```

Figure 5.22: Compiling the Model

5.4.8. Callback's:

To intervene in the model during the learning process, the callbacks shown in Figure *5.23* have been added.

ReduceLROnPlateau: It is added to reduce the learning rate when the validation loss does not improve, so that as we approach the best parameters, the learning rate is reduced to avoid skipping this value.

Factor => Determines the rate at which the Learning Rate will be reduced. For example, if it is 0.1, the learning rate will be reduced in 10% intervals.

Patience => Determines how long to wait before the learning rate is reduced. If there is no improvement for the specified number of epochs, the learning rate is reduced.

Min_delta => It sets the threshold value used to measure the best result. So, if val_loss shows less improvement than this value, it is not considered an improvement.

Cooldown => It determines the number of epochs to wait for resume the process after the learning rate has been reduced.

Min_lr => Determines the minimum value which the learning rate can fall.

EarlyStopping: Stops the training process and restores optimal weights when loss of verification does not improve over a period.

Patience => As explained above, this determines how many epochs to wait before stopping training.

Restore_best_weights => It determines whether the best weights are saved when training is stopped. In these studies, the best weights were always saved.

```
#Decreasing Learning Rate
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(
   monitor='val_loss',
   factor=0.1,
    patience=7,
    verbose=1,              # Mesajları yazdır
    mode='auto',
    min_delta=0.0001,       # bu deger kadar degisim olmazsa
    cooldown=0,
    min_lr=1e-7
)

# EarlyStopping callback'ini tanımlayın
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=15,
    verbose=1,              # Mesajları yazdır
    mode='auto',
    restore_best_weights=True
)
history = model.fit(
    train_ds,
    batch_size=BATCH_SIZE,
    validation_data=val_ds,
    verbose=1,
    epochs=EPOCHS,
    callbacks=[early_stopping, reduce_lr]
)
```

Figure 5.23: Callbacks.

Then, as shown in Figure *5.24*, the model was trained with different batch sizes and different epochs using train and validation data sets and callbacks. The model was tested on the test data structure after training.

Batch size => The amount of data to be processed in each step is determined by the batch size. At the same time, more data can be used for training with larger batches and therefore each iteration is also less frequent hence improved learning of generic features. Smaller batch

sizes allow training on less data at the same time and more frequent iterations, making it more sensitive to noise in case of erroneous data. Larger batch sizes require more memory and processing power, so the hardware may be insufficient, and training may fail. In this project, many batch sizes from 8 to 88 were tried. According to the information obtained from these experiments, if the hardware is powerful and sufficient, increasing the batch size plays an important role in faster training completion.

Epochs => When training a model, "number of epochs" is how many times the model is trained with the full dataset. Overfitting might happen if you train too many times whereas underfitting when training too fewer times. For this reason, in this project, It is kept the number of epochs high and added callbacks such as Early Stopping and Learning Rate Reduction. Thus, the training is made more accurate and healthier.

```
history = model.fit(
    train_ds,
    batch_size=BATCH_SIZE,
    validation_data=val_ds,
    verbose=1,
    epochs=EPOCHS,
    callbacks=[early_stopping, reduce_lr]
)

scores = model.evaluate(test_ds)
```

Figure 5.24: Training of The Model

5. Model Performance Evaluation

After training, the model was evaluated with the test data structure. After the test, the accuracy and loss changes of the model were printed on the screen as a graph with the code in Figure *5.25*. Thus, the learning status of the model was clearly understood.

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

#graphs for accuracy and loss of training and validation data
plt.figure(figsize = (20,20))
plt.subplot(2,3,1)
plt.plot(range(len(acc)), acc, label = 'Training Accuracy')
plt.plot(range(len(val_acc)), val_acc, label = 'Validation Accuracy')
plt.legend(loc = 'lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2,3,2)
plt.plot(range(len(loss)), loss, label = 'Training Loss')
plt.plot(range(len(val_loss)), val_loss, label = 'Validation Loss')
plt.legend(loc = 'upper right')
plt.title('Training and Validation Loss')
```

Figure 5.25: Accuracy and Loss Graph Code

Then, the actual label and prediction results and confidence rate of the model with 7-8 image samples were printed on the screen with the code shown in the Figure *5.26*. After that, the well-trained model was saved as .H5 file.

```
def predict(model, img):
    img_array = tf.keras.preprocessing.image.img_to_array(images[i].numpy())
    img_array = tf.expand_dims(img_array, 0)

    predictions = model.predict(img_array)

    predicted_class = class_names[np.argmax(predictions[0])]
    confidence = round(100 * (np.max(predictions[0])), 2)
    return predicted_class, confidence

"""**Now run inference on few sample images**"""

plt.figure(figsize=(15, 20))
for images, labels in test_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))

        predicted_class, confidence = predict(model, images[i].numpy())
        actual_class = class_names[labels[i]]

        plt.title(f"Actual: {actual_class},\n Predicted: {predicted_class}.\n Confidence: {confidence}%")

        plt.axis("off")
```

Figure 5.26: Confidence, Prediction and Actual Label of a few Images

The trained and saved model was tested on a test data set consisting of 200 images that it had not seen before. Thus, the real-world performance of the model was evaluated.
According to all trial results, the highest accuracy rate was observed as 99.7% for Training and 85.5% for Test, the accuracy rate of the test dataset varies between 83%-85% on average.

6. Model Integration

In this step, the raspberry pi operating system was first installed, and the device was made operational. Then the H5 file of the model and the data were uploaded to the raspberry pi. The libraries required for the code to run were downloaded and installed and the code and the device were made to run smoothly. Then the test of the model was performed on Raspberry pi (Figure *5.27*).
In this step, it was aimed to create a low-cost and portable system. It was found that a system capable of mobile simultaneous ochratoxin detection could be developed in the future.

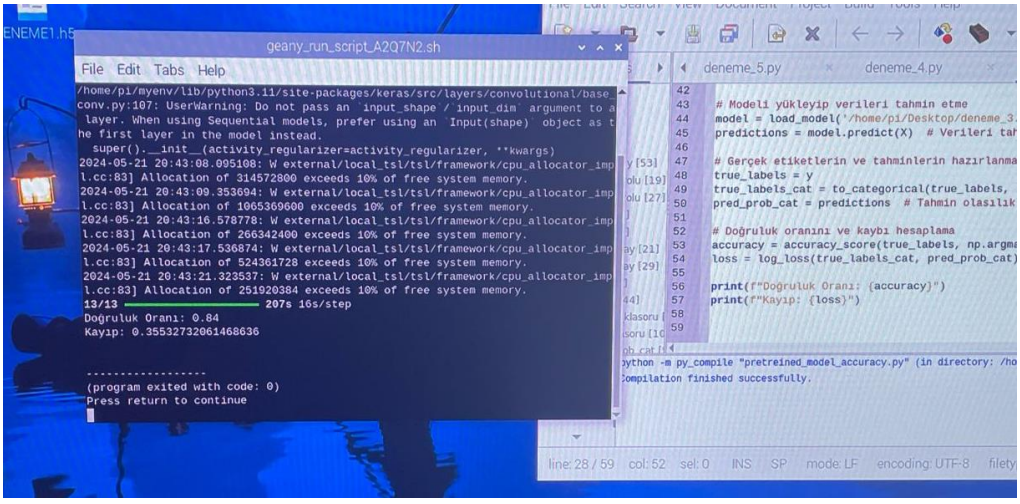

Figure 5.27: Raspberry Pi Integration and Test

6. RESULTS AND DISCUSSION

In this project, many different pretrained models such as VGG, Resnet50, Efficinet, XGBoost and a special CNN model created from scratch were tested. As a result of the experiments, CNN model shows the best perormance and gives the best results. Different optimizers, hyperparameter settings and layers were tried to improve the accuracy and consistency of the CNN model. The results of the experiments will be evaluated below.

	Traning		Test
	Accuracy	Loss	Accuracy
Trial1	%85.5	%40.8	-
Trial2	%74.5	%51.0	-
Trial3	%87.0	%30.0	-
Trial4	%86.87	%34.4	-
Trial5	%90.0	%34.5	-
Trial6	%68.1	%63.9	-
Trial7	%71.5	%55.6	-
Trial8	%97.0	%27.0	%85.5
Trial9	%94.5	%27.2	-
Trial10	%97.92	%12.56	%81.5
Trial11	%94.53	%27.74	%84.0
Trial12	%94.53	%31.96	-
Trial13	%94.00	%31.11	%82.0
Trial14	%97.73	%17.94	%84.5
Trial15	%97.86	%18.21	%78.0
Trial16	%94.27	%47.48	%83.75
Trial17	%92.5	%23.71	%82.5
Trial18	%97.66	%17.96	%82.25
Trial19	%93.75	%27.18	%82.75
Trial20	%96.09	%20.82	%82.5
Trial21	%96.09	%18.94	%85.0
Trial22	%96.35	%21.8	%83.75
Trial23	%92.19	%23.05	%80.5
Trial24	%99.0	%13.0	-
Trial25	%88.0	%34.0	-
Trial26	%91.77	%26.97	-
Trial27	%93.0	%29.3	-
Trial28	%89.48	%36.48	-
Trial29	%90.73	%30.2	-
Trial30	%89.38	%27.67	-
Trial31	%92.81	%21.21	-
Trial32	%94.53	%41.16	%85.5
Trial33	%93.23	%22.08	%83.75
Trial34	%92.71	%23.67	%83.75
Trial35	%93.23	%29.31	%82.25
Trial36	%95.31	%19.22	%82.5
Trial37	%96.88	%12.64	%85.5

6.1. Different Trials

6.1.1. VGG16

Trial1: Pure Code Trainable:False, Include top: False

Loss: 0.408, Accuracy: %85.5

Trial2: With Blurriness, Data Augmentation and Early Stopping, Trainable : False, Include top: False

Loss: 0.51, Accuracy: %74.5

Trial3: Additional Layers Added, Trainable: True, Include Top: False

Loss: 0.30, Accuracy: %87

Trial4: Additional Layers Added, With Blurriness, Dropout, Early Stopping, Trainable: True, Include Top: False

loss: 0.344, Accuracy: %86.87

6.1.2. LSTM

Trial5: With Cross Validation, Dropout and Early Stopping, 40 Epoch

Loss: 0.345, Accuracy: %90

Trial6: With Early Stopping and Dropout, 40 Epoch

Loss: 0.639, Accuracy: %68.1

Trial7: With Early Stopping, Blurriness, Dropout and Cross Validation

Loss: 0.556, Accuracy: %71,5

6.1.3. CNN MODEL

In this section, trainings were made with visuals with Absorbance and Reflectance values in a single graph. There is no Reduced Learning Rate in this section.

Trial8: Firstly, training was performed with Batch Size (32), Early Stopping (Patience=10) => 72 Epochs, Dropout (0.5), L2Regularizer (0.001) and Learning Rate = 0.001 settings with the Dataset divided into 80% Train, 10% Validation and 20% Test sets. Training Results are shown in Figure *6.1*. As a result of this training, 27% Training Loss and 97% Training Accuracy were obtained. In unseen tests, 85.5% Accuracy was obtained.

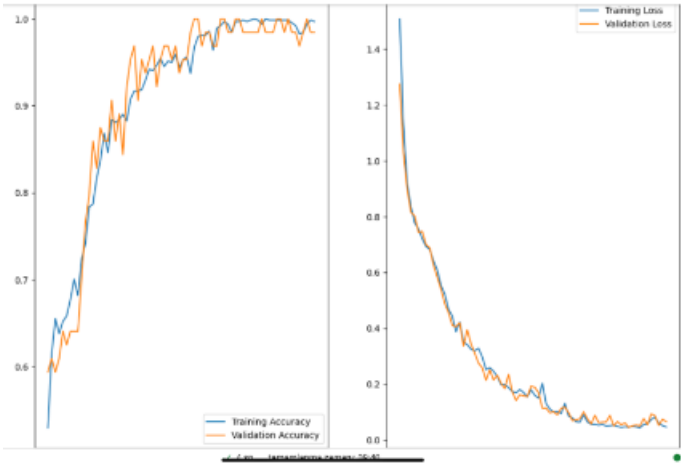

Figure 6.1: Loss and Accuracy Graph

Trial9: In another experiment, Batch Size (32), Early Stopping (Patience=10) => 56 Epochs, NO Dropout, L2Regularizer (0.001) and Learning Rate = 0.001 settings, the Dataset was divided into 80% Train, 10% Validation and 10% Test sets. Training Results are shown in Figure *6.2*. As a result of this training, 27.2% Training Loss and 94.5% Training Accuracy were obtained.

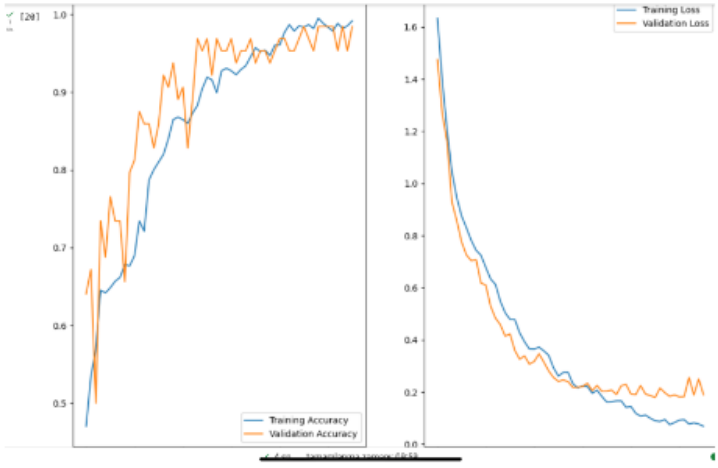

Figure 6.2: Accuracy and Loss Graph

Trial10: In another experiment, Batch Size (32), Early Stopping (Patience=10) => 48 Epochs, Dropout (0.5), L2Regularizer (0.001) and Learning Rate = 0.0001 settings, the Dataset was divided into 80% Train, 10% Validation and 10% Test sets. Training Results are shown in Figure

6.3. As a result of this training, 12.56% Training Loss and 97.92% Training Accuracy were obtained. In unseen tests, 81.5% Accuracy was obtained.

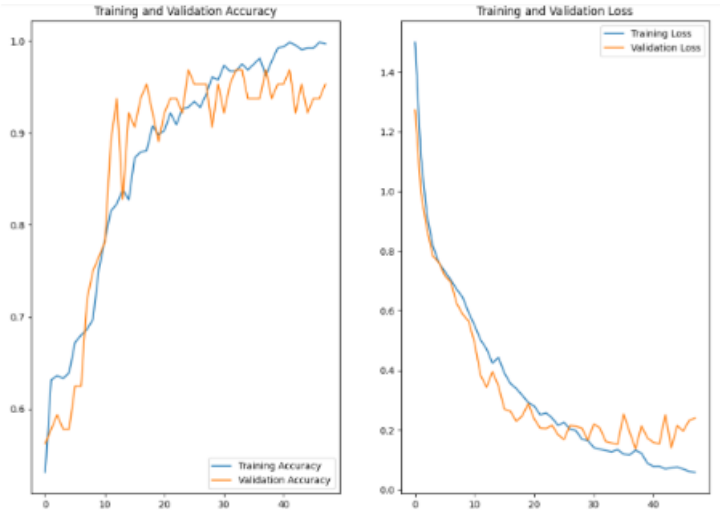

Figure 6.3: Accuracy and Loss Graph

Trial11: In another experiment, Batch Size (64), Early Stopping (Patience=10) => 58 Epochs, Dropout (0.5), L2Regularizer (0.001) and Learning Rate = 0.0001 settings, the Dataset was divided into 80% Train, 10% Validation and 10% Test sets. Training Results are shown in Figure *6.4*. As a result of this training, 27.74% Training Loss and 94.53% Training Accuracy were obtained. In unseen tests, 84% Accuracy was obtained.

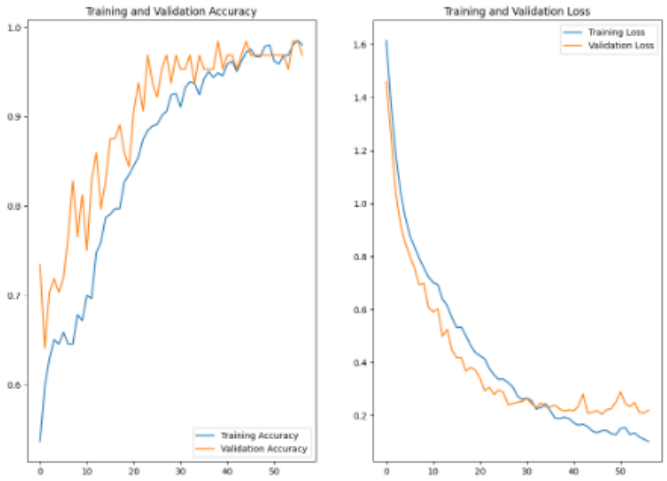

Figure 6.4: Accuracy and Loss Graph

Trial12: In another experiment, Batch Size (64), Early Stopping (Patience=10) => 69 Epochs, Dropout (0.5), L2Regularizer (0.001) and Learning Rate = 0.00005 settings, the Dataset was divided into 80% Train, 10% Validation and 10% Test sets. Training Results are shown in Figure

6.5. As a result of this training, 31.96% Training Loss and 94.53% Training Accuracy were obtained.

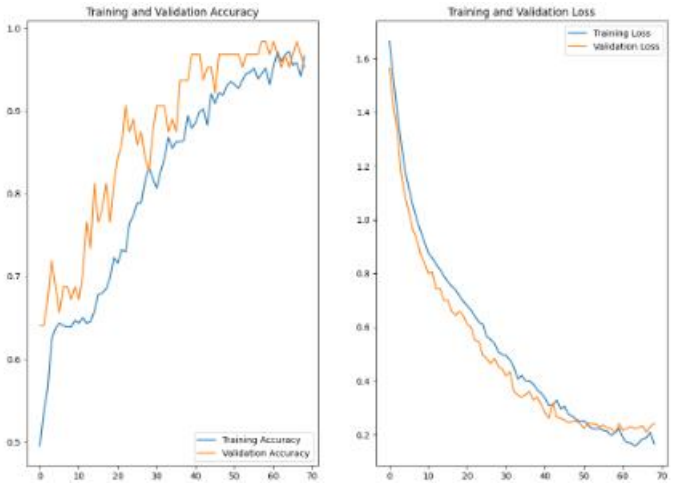

Figure 6.5: Accuracy and Loss Graph

Trial13: In another experiment, Batch Size (80), Early Stopping (Patience=10) => 58 Epochs, Dropout (0.5), L2Regularizer (0.001) and Learning Rate = 0.00001 settings, the Dataset was divided into 80% Train, 10% Validation and 10% Test sets. Training Results are shown in Figure *6.6*. As a result of this training, 31.11% Training Loss and 95.00% Training Accuracy were obtained. In unseen tests, 82% Accuracy was obtained.

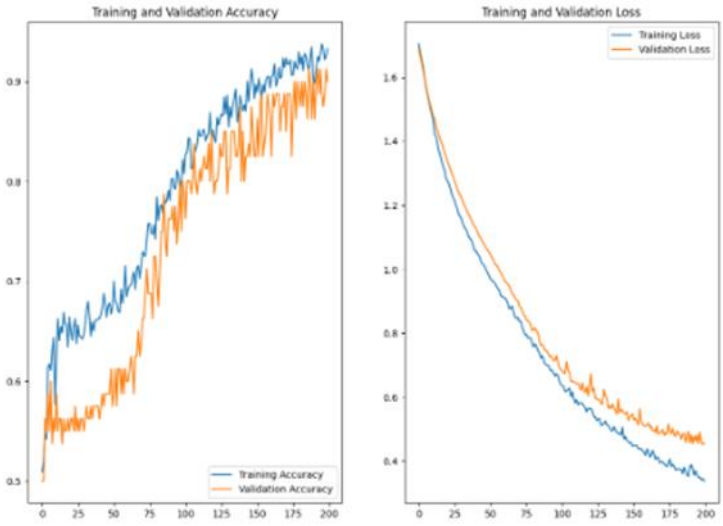

Figure 6.6: Accuracy and Loss Graph

Trial14: In another experiment, Batch Size (88), Early Stopping (Patience=10) => 130 Epochs, Dropout (0.5), L2Regularizer (0.001) and Learning Rate = 0.00003 settings, the Dataset was

divided into 80% Train, 10% Validation and 10% Test sets. Training Results are shown in Figure *6.7*. As a result of this training, 17.94% Training Loss and 97.73% Training Accuracy were obtained. In unseen tests, 84.5% Accuracy was obtained.

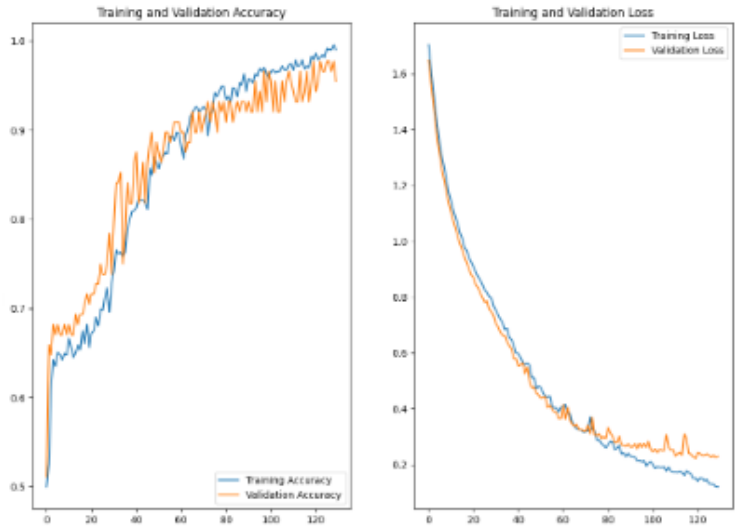

Figure 6.7: Accuracy and Loss Graph

Trial15: In another experiment, Batch Size (70), Early Stopping (Patience=10) => 84 Epochs, Dropout (0.5), L2Regularizer (0.001) and Learning Rate = 0.0001 settings, the Dataset was divided into 80% Train, 10% Validation and 10% Test sets. Training Results are shown in Figure *6.8* As a result of this training, 18.21% Training Loss and 97.86% Training Accuracy were obtained. In unseen tests, 78% Accuracy was obtained.

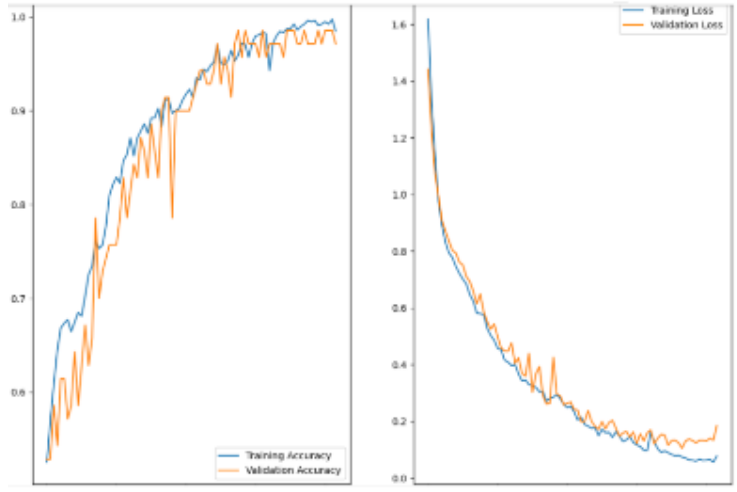

Figure 6.8: Accuracy and Loss Graph

Trial16: In another experiment, Batch Size (64), Early Stopping (Patience=10) => 84 Epochs, Dropout (0.5), L2Regularizer (0.001) and Learning Rate = 0.0001 settings, the Dataset was

divided into 80% Train, 10% Validation and 10% Test sets. Training Results are shown in Figure *6.9* As a result of this training, 47.48% Training Loss and 94.27% Training Accuracy were obtained. In unseen tests, 83.75% Accuracy was obtained.

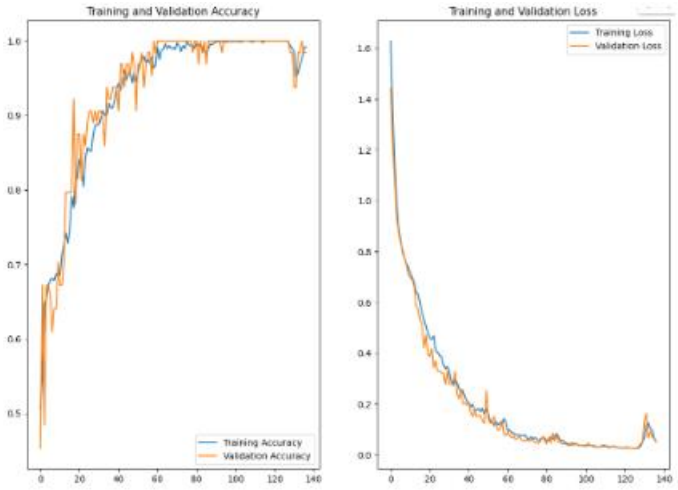

Figure 6.9: Accuracy and Loss Graph

Trial17: In another experiment, Batch Size (80), Dropout (0.5), Early Stopping (Patience=10) => 104 Epochs, L2Regularizer (0.001) and Learning Rate = 0.00003 settings, the Dataset was divided into 80% Train, 10% Validation and 10% Test sets. Non-Restrictive graph images used. Training Results are shown in Figure *6.10* As a result of this training, 23.71% Training Loss and 92.50% Training Accuracy were obtained. In unseen tests, 82.5% Accuracy was obtained.

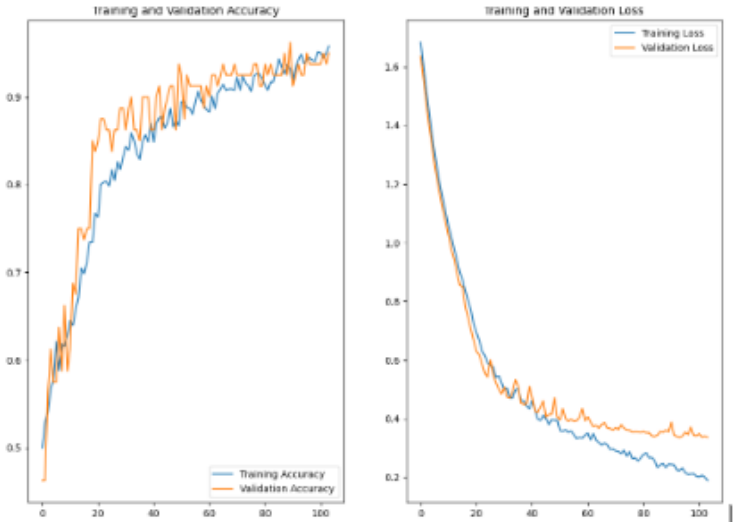

Figure 6.10: Accuracy and Loss Graph

Trial18: In another experiment, Batch Size (32), Dropout (0.5), Early Stopping (Patience=8) => 68 Epochs, L2Regularizer (0.001) and Learning Rate = 0.0001 settings, the Dataset was divided into 80% Train, 10% Validation and 10% Test sets. Non-Restrictive graph images used. Training Results are shown in Figure *6.11*. As a result of this training, 17.96% Training Loss and 97.66% Training Accuracy were obtained. In unseen tests, 82.25% Accuracy was obtained.

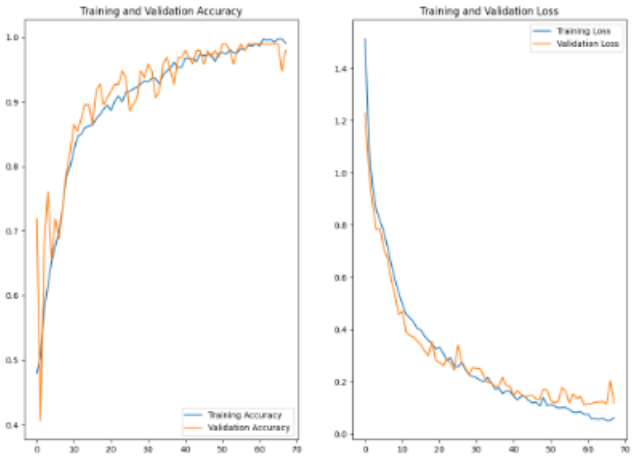

Figure 6.11: Accuracy and Loss Graph

Trial19: In another experiment, Batch Size (80), Dropout (0.5), Early Stopping (Patience=10) => 97 Epochs, L2Regularizer (0.001) and Learning Rate = 0.00003 settings, the Dataset was divided into 80% Train, 10% Validation and 10% Test sets. Limited graph images used. Training Results are shown in Figure *6.12*. As a result of this training, 27.18% Training Loss and 93.75% Training Accuracy were obtained. In unseen tests, 82.75% Accuracy was obtained.

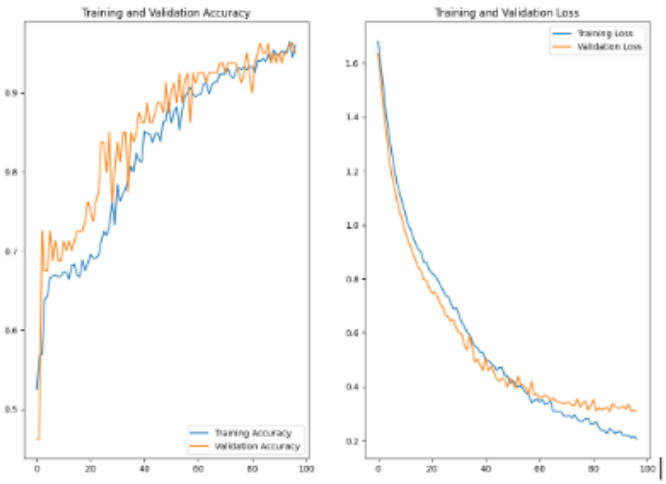

Figure 6.12: Accuracy and Loss Graph

Trial20: In another experiment, Batch Size (32), Dropout (0.5), Early Stopping (Patience=8) => 48 Epochs, L2Regularizer (0.001) and Learning Rate = 0.0001 settings, the Dataset was divided into 80% Train, 10% Validation and 10% Test sets. Limited graph images and Softmax activation function used. Training Results are shown in Figure *6.13*. As a result of this training, 20.82% Training Loss and 96.09% Training Accuracy were obtained. In unseen tests, 82.5% Accuracy was obtained.

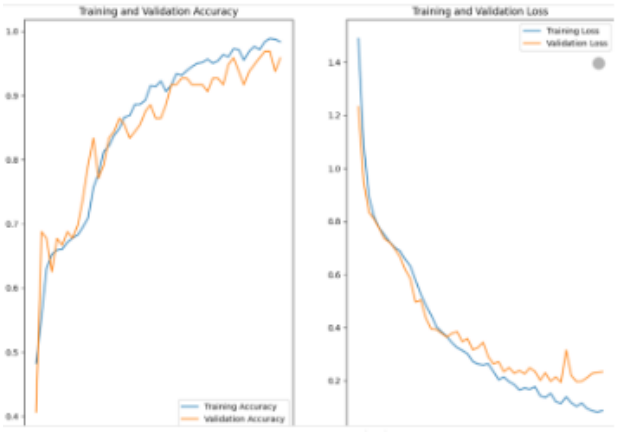

Figure 6.13: Accuracy and Loss Graph

Trial21: In another experiment, Batch Size (32), Dropout (0.5), Early Stopping (Patience=8) => 48 Epochs, L2Regularizer (0.001) and Learning Rate = 0.0001 settings, the Dataset was divided into 80% Train, 10% Validation and 10% Test sets. Limited graph images and Sigmoid activation function used. Training Results are shown in Figure *6.14*. As a result of this training, 18.94% Training Loss and 96.09% Training Accuracy were obtained. In unseen tests, 85% Accuracy was obtained.

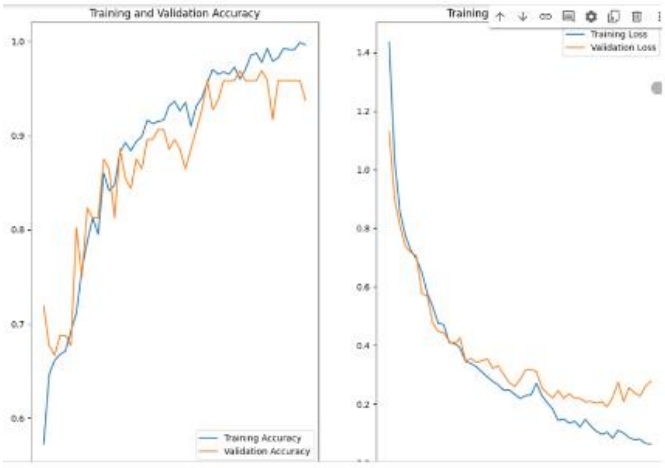

Figure 6.14: Accuracy and Loss Graph

Trial22: In another experiment, Batch Size (64), Dropout (0.5) + Dropout (0.8), Early Stopping (Patience=8) => 78 Epochs, L2Regularizer (0.001) and Learning Rate = 0.0001 settings, the Dataset was divided into 80% Train, 10% Validation and 10% Test sets. Limited graph images and Sigmoid activation function used. Training Results are shown in Figure *6.15*. As a result of this training, 21.80% Training Loss and 96.35% Training Accuracy were obtained. In unseen tests, 83.75% Accuracy was obtained.

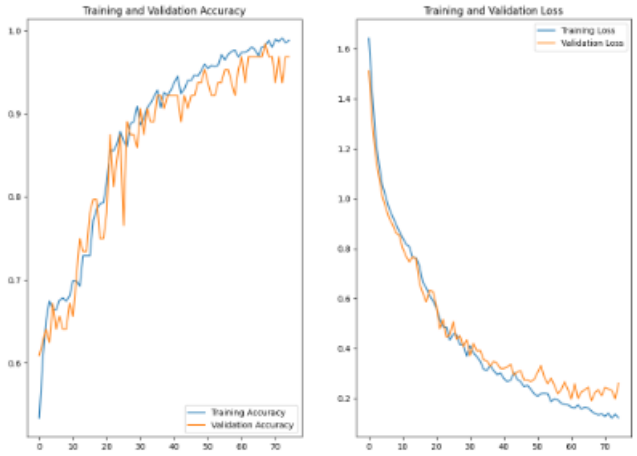

Figure 6.15: Accuracy and Loss Graph

Trial23: In another experiment, Batch Size (64), Dropout (0.5), Early Stopping (Patience=10) => 53 Epochs, L2Regularizer (0.001) and Learning Rate = 0.0001 settings, the Dataset was divided into 80% Train, 10% Validation and 10% Test sets. Only Absorbance graph images used. Training Results are shown in Figure *6.16*. As a result of this training, 23.05% Training Loss and 92.19% Training Accuracy were obtained. In unseen tests, 80.5% Accuracy was obtained.

Figure 6.16: Accuracy and Loss Graph

Trial24: After these experiments, learning rate reduction was added to the code and experiments were made in this way. The settings of the callbacks are as in Figure *6.19*. The following experiments were done with 32 Batch Size, learning rate = 0.0001, Early stopping added. The result in Figure *6.17* was done by dividing the data set into 80% Training, 10% Validation and 10% Test and trained with non-Limited graphs. Sigmoid activation function was used. 99% Train accuracy and 13% Loss was achieved.

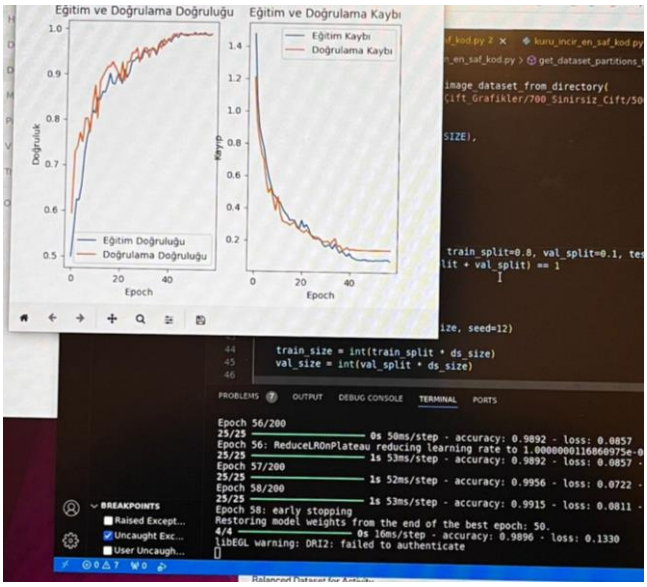

Figure 6.17: %80 Training, %10 Validation, %10 Test

Trial25: With the same settings as above, only the data set was split into 70% Train, 20% Validation and 10% Test. The result is 88% Accuracy and 34% Loss and the graph of the accuracy and loss is shown in Figure *6.18*.

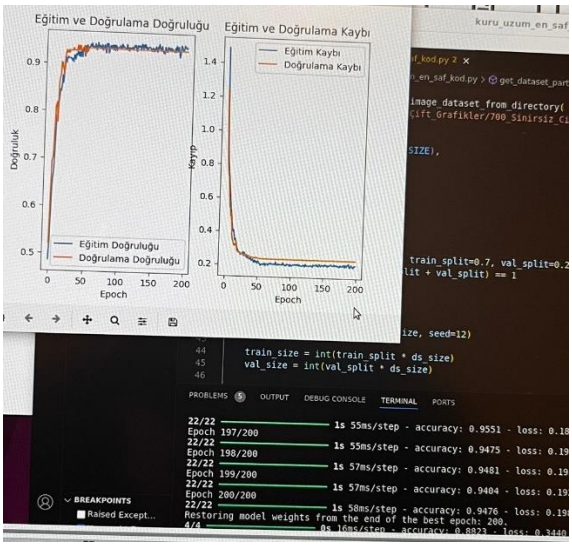

Figure 6.18: %70 Train, %20 Validation, %10 Test Dataset Separation

```
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(
    monitor='val_loss',     # Validation loss'u izle
    factor=0.1,             # Öğrenme oranını %10'a düşür
    patience=3,             # 5 epoch boyunca iyileşme olmazsa
    verbose=1,              # Mesajları yazdır
    mode='auto',            # 'min', 'max', 'auto'
    min_delta=0.0001,       # Yeni optimum için eşik
    cooldown=0,             # Öğrenme oranı düşürüldükten sonra kaç epoch bekle
    min_lr=1e-8             # Öğrenme oranının düşebileceği minimum değer
)

# EarlyStopping callback'ini tanımlayın
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',     # Validation loss'u izle
    patience=8,             # 10 epoch boyunca iyileşme olmazsa eğitimi durdur
    verbose=1,              # Mesajları yazdır
    mode='auto',            # 'min', 'max', 'auto'
    restore_best_weights=True  # En iyi ağırlıkları geri yükle  ,reduce_lr,
)
history = model.fit(
```

Figure 6.19: Early Stopping and Learning Rate Reduction

In the Figure *6.20*, Figure *6.21*, Figure *6.22*, Figure *6.23* and Figure *6.24*, different combinations of the parameters of the ReduceLROnPlateau function were experimented with and the results are shown in the figures. For all these experiments, all other settings in Figure *6.17* were kept constant and only the settings of the Learning rate reduction function were changed. If we look at the results obtained from these experiments.

Trial26: In Figure *6.20*, the min_lr was set to 1e^-10, and Train Accuracy was 91.77% and Loss was 26.97.

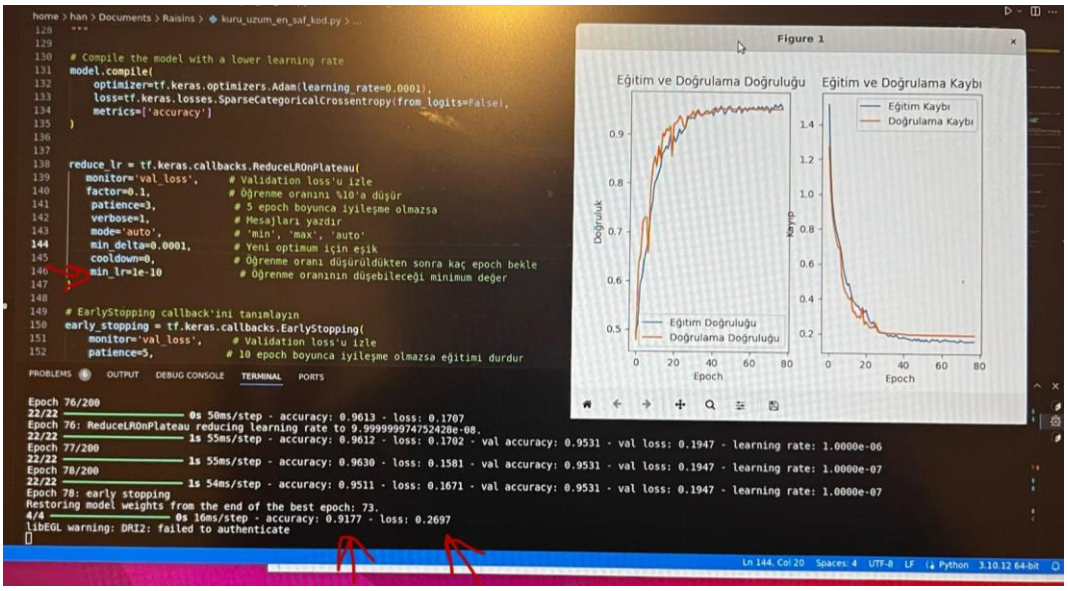

Figure 6.20: ReduceLR => min_lr=1e^-10

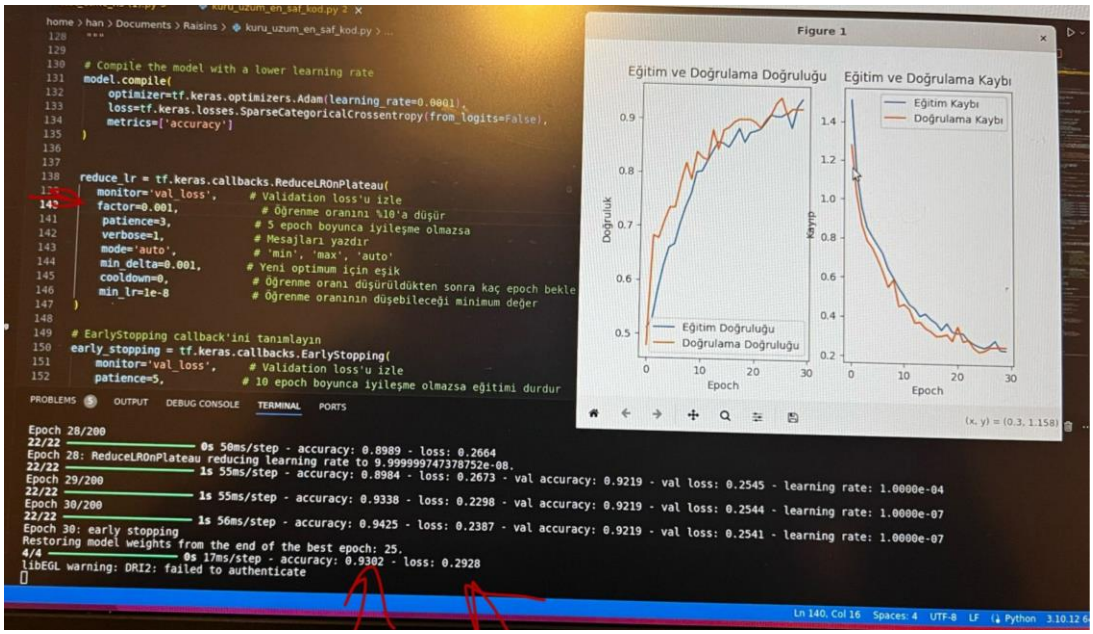

Figure 6.21: Reduce_LR=> factor=0.001

Trial27: In Figure *6.21* the factor was set to 0.001 and Train accuracy was %93, and Loss was %29.3.

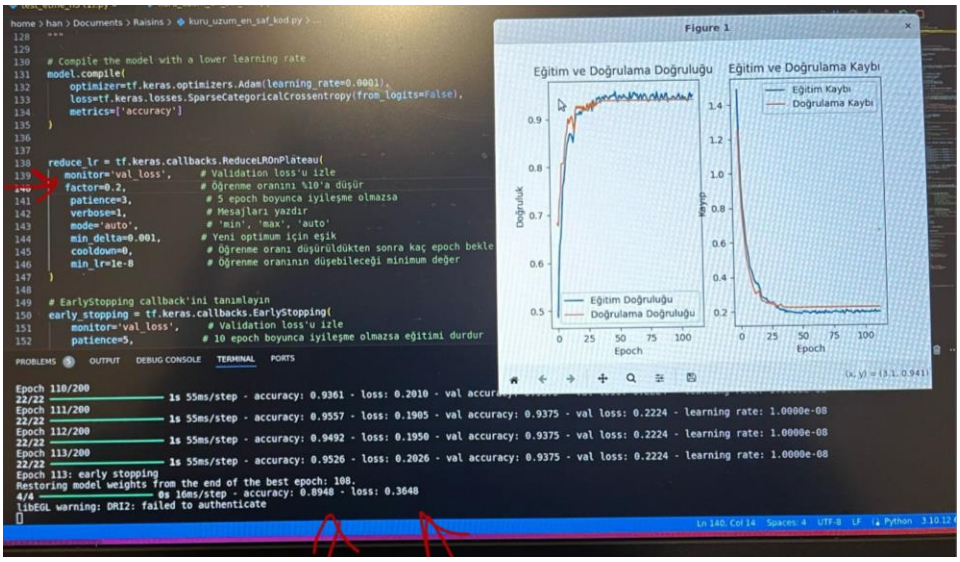

Figure 6.22: Reduce_LR => factor=0.2

Trial28: In Figure *6.22* the factor was set to 0.2 and Train accuracy was %89.48, and Loss was %36.48.

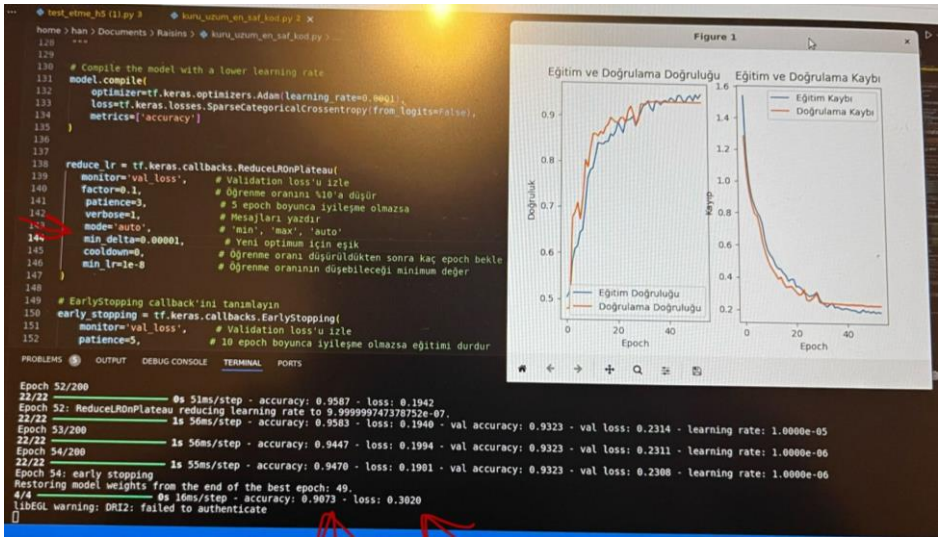

Figure 6.23: Reduce_LR => min_delta=0.00001

Trial29: In Figure *6.23*, the min_delta was set to 0.00001 and Train accuracy was %90.73, and Loss was %30.2.

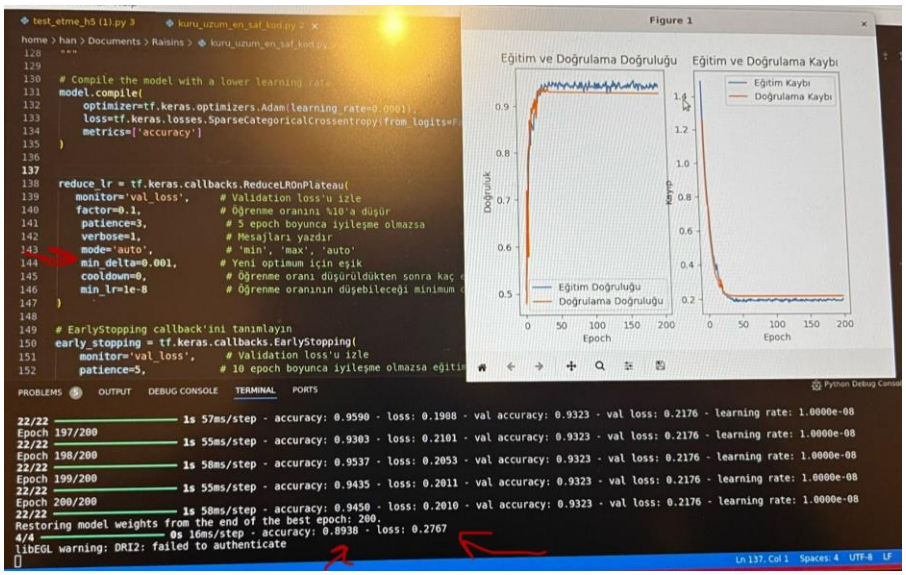

Figure 6.24: Reduce_LR => min_delta=0.001

Trial30: In Figure *6.24* the min_delta was set to 0.001 and Train accuracy was %89.38, and Loss was %27.67.

Trial31: In addition to these experiments, the Cross Validation function was added to the code and k_fold = 5, with all other settings remaining the same. Figure *6.25* shows the results of the code with cross validation added. As a result of this experiment, Training accuracy was 92.81% and Loss was 21.21%.

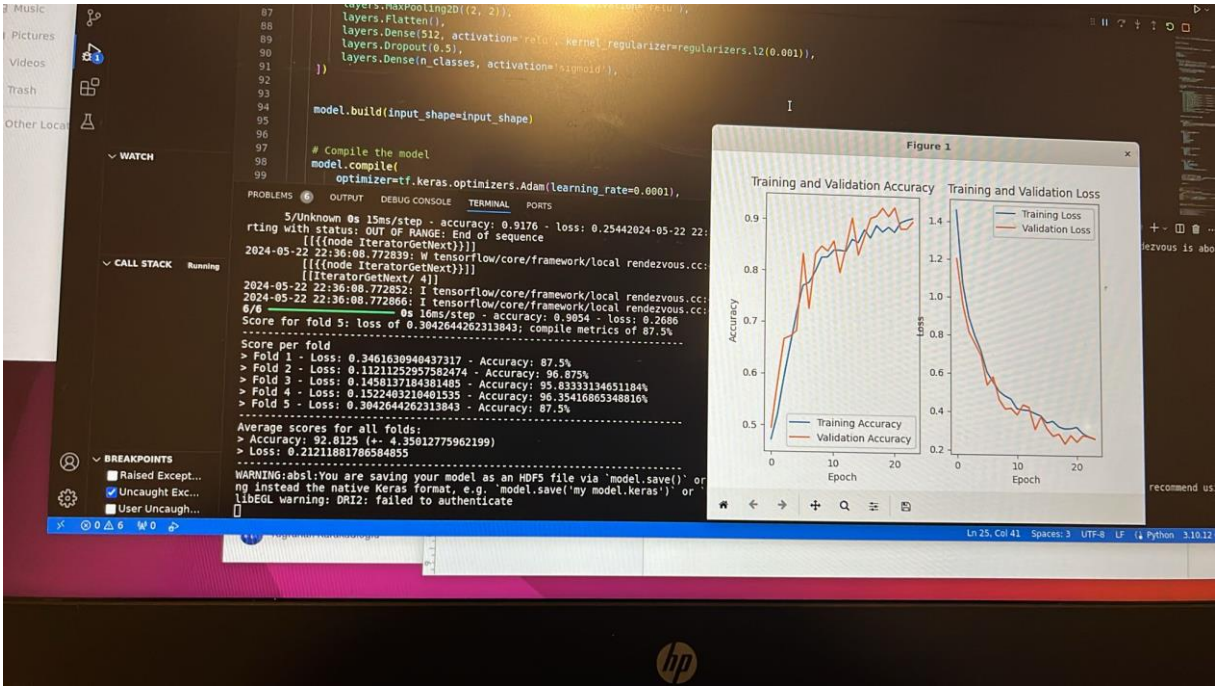

Figure 6.25: CNN With Cross Validation

Trial32: In the Figure *6.26* , the dense layers in the model have been increased and Dropout has been added between each layer. This attempt gives loss: 0.4116 - accuracy: 0.9453 at training stage, after model saving it gives Accuracy Rate: 0.855.

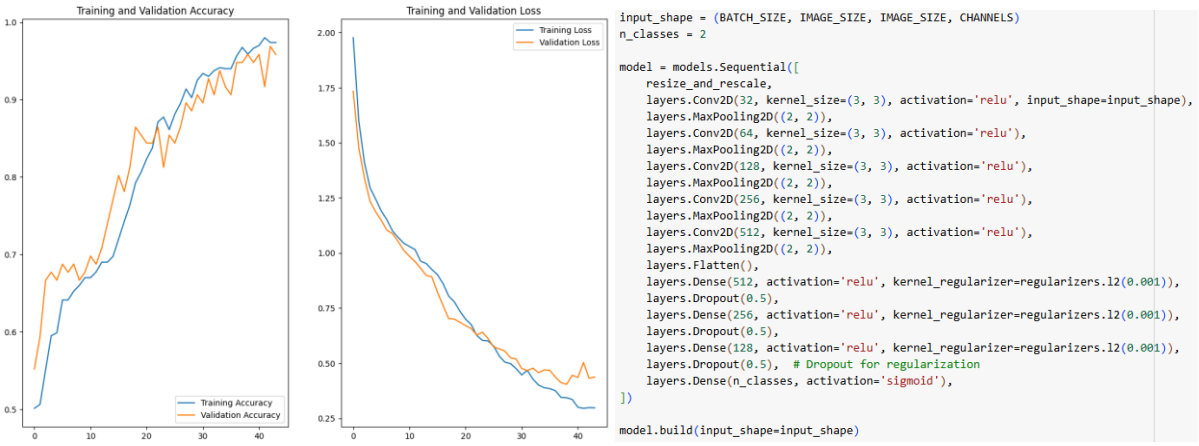

Figure 6.26: Increased Dense Layers

Trial33: In the Figure 6.27 below, 64 Batch size applied, Unlimited dataset was used, and Train dataset divided by 0.7 Train, 0.2 Validation and 0.1 Test and Data augmentation with Random brightness was applied, Regularizers applied as 0.7 dropout and 0.001 L2, Adam optimizer with 0.00001 Learning rate was applied and Early stopping was active. Sigmoid activation function applied. The result was reached as %93.23 Training accuracy, %22.08 Training loss and with unseen test %83.75 Test accuracy reached.

Figure 6.27: CNN Result with Unlimited Dataset

Trial34: In the Figure 6.28 below, 64 Batch size applied, Unlimited dataset was used, and Train dataset divided by 0.7 Train, 0.2 Validation and 0.1 Test and Data augmentation with Random brightness was applied, Regularizers applied as 0.8 dropout and 0.001 L2, Adam optimizer with 0.00001 Learning rate was applied and Early stopping was active. Sigmoid activation function

applied. The result was reached as %92.71 Training accuracy, %23.67 Training loss and with unseen test %83.75 Test accuracy reached.

Figure 6.28: CNN Result with Unlimited Dataset

Trial35: In the Figure 6.29 below, 64 Batch size applied, Unlimited dataset was used and Train dataset divided by 0.7 Train, 0.2 Validation and 0.1 Test and Data augmentation with Random brightness was applied, Regularizers applied as 0.8 dropout and 0.001 L2, Adam optimizer with 0.00001 Learning rate was applied but Reduce LR and Early stopping was active. Sigmoid activation function applied. The result was reached as %93.23 Training accuracy, %29.31 Training loss and with unseen test %82.25 Test accuracy reached.

Figure 6.29: CNN Result with Unlimited Dataset

Trial36: In the Figure *6.30* below, 64 Batch size applied, Unlimited dataset was used and Train dataset divided by 0.7 Train, 0.2 Validation and 0.1 Test and Data augmentation with Random brightness was applied, Regularizers applied as 0.8 dropout and 0.001 L2, Adam optimizer with 0.0001 Learning rate was applied and Early stopping was active. Sigmoid activation function applied. The result was reached as %95.31 Training accuracy, %19.22 Training loss and with unseen test %82.5 Test accuracy reached.

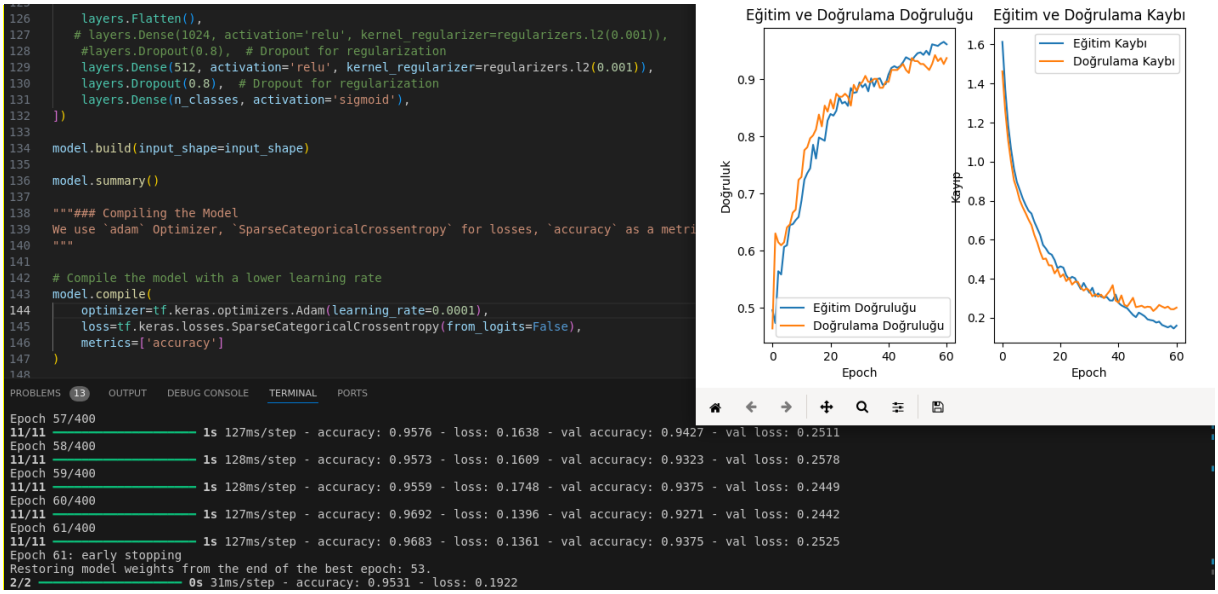

Figure 6.30: CNN with Unlimited Dataset

Trial37: In the Figure 6.31 below, 64 Batch size applied, Unlimited dataset was used and Train dataset divided by 0.7 Train, 0.2 Validation and 0.1 Test and Data augmentation with Random brightness was applied, Regularizer applied as 0.001 L2, Adam optimizer with 0.0001 Learning rate was applied and Early stopping was active. Sigmoid activation function applied. The result was reached as %96.88 Training accuracy, %12.64 Training loss and with unseen test %85.5 Test accuracy reached.

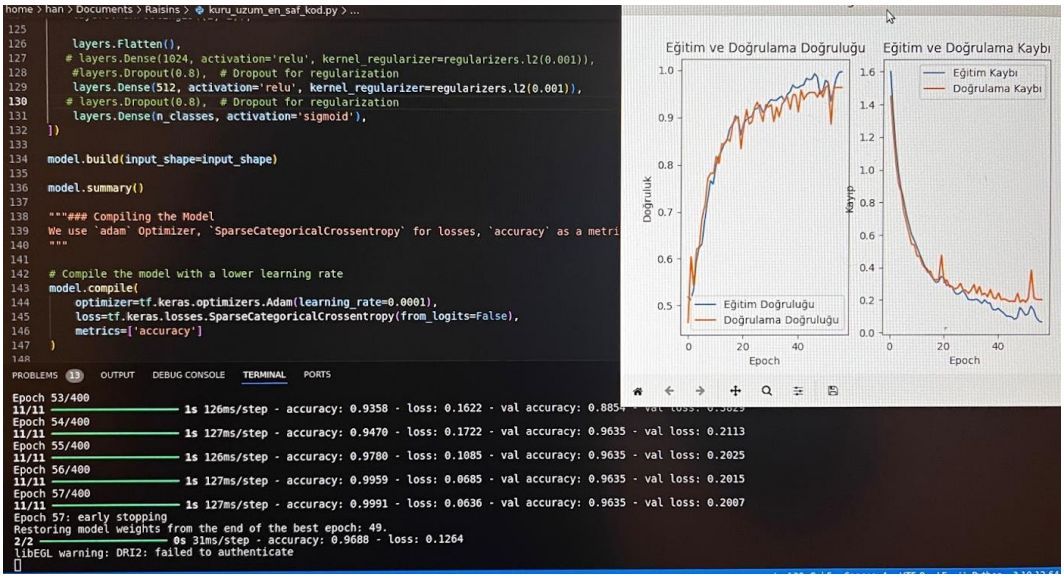

Figure 6.31: CNN Result with Unlimited Dataset

As observed at the general flow in these experiments, as expected, accuracy shows a graph that increases regularly, and loss decreases regularly in almost all experiments. There are some fluctuations, but this is the general opinion.

When the graphs obtained after training are analyzed, a general upward trend is observed in both training accuracy and validation accuracy. It can be confirmed that the capacity of the model to learn and generalize increases in time. However, the fluctuations in the graphs may be due to suboptimal conditions during the measurement and inadequacies of the enclosure designed for the measurement device.

In the loss and loss of validation graphs, a simultaneous decrease is observed, indicating the effectiveness of the model and the training process. However, in some trials in the constrained graphs, the difference between validation loss and loss starts to increase towards the end of the training. However, this is much less in the unconstrained graphs and loss and validation loss follow very similar paths. The lack of a sudden spike in accuracy and validation accuracy is due to the careful selection of optimizers, parameters, hyperparameters and regularizers. This shows that the training process was carried out in a stable and controlled manner.

When the saved model was saved in h5 format and tested on a separate dataset that the model did not see, a slightly higher loss was observed compared to the training data. This is a common result of testing the model with test data that it has never encountered before. The model performed well as evidenced by accuracy values ranging between 83-86 percent on the test dataset and 96-98 percent on training dataset.

The findings indicate success and promise further studies in this domain, as confirmed by the rates of accuracy and loss for this project The study proved the effectiveness of the combination of NIR Spectrometry and artificial intelligence in detecting OTA contamination in raisins and revealed that these methods have broad application potential in the fields of agriculture and food safety. In the detection of ochratoxin in other agricultural products, the development of mobile and portable versions in future research might be considered. In this way, safer and higher quality agricultural products can be obtained and thus contribute to human health.

7. CONCLUSION

In this study, spectral analysis and machine learning techniques were employed to classify raisins contaminated with ochratoxin and healthy ones. The findings and recommendations from this analysis are outlined below.

The results indicate a significant difference in the spectral data of raisins contaminated with ochratoxin compared to healthy ones. Particularly, lower absorbance rates were observed in raisins contaminated with ochratoxin, suggesting that spectral analysis could be an effective method for ochratoxin detection.

Tests conducted on machine learning models revealed that a specially designed CNN model achieved the highest accuracy. These results suggest that custom-designed models may outperform standard models in specific problem domains.

During the implementation phase, the successfully trained model was integrated into a Raspberry Pi board. This demonstrates the feasibility of using a low-cost, portable device for ochratoxin detection in field conditions.

For future research, it is recommended to utilize larger-scale datasets and compare different machine learning models. Additionally, exploring the applicability of spectral analysis to other food items and conducting further tests in field conditions would be beneficial.

The findings of this study represent a significant step in food safety and may contribute to the development of new and effective methods for ochratoxin detection in the industry.

Introduction: This chapter addresses the significant issue of ochratoxin A (OTA) contamination in raisins, particularly within the Turkish raisin industry. OTA is a mycotoxin that's harmful in people's bodies. it is produced by molds like Aspergillus and Penicillium. This leads to severe health dangers such as risk of developing cancer! The introduction highlights the economic and health implications of OTA contamination, emphasizing the importance of developing efficient detection methods.

The introduction discusses existing detection methods like HPLC and ELISA, highlighting their drawbacks in terms of time, cost, and dependability. The introduction explains existing techniques to detect hCG, e.g., HPLC or ELISA, and their drawbacks including duration,

expenses, and dependability. It follows up with a statement about how the project will involve finding a faster yet more certain way of identifying hCG using Near-Infrared (NIR) Spectrometer and AI. This new approach is expected to provide real-time, accurate detection of OTA in raisins, improving product quality and safety.

Research Objective: The study in this chapter will focus specifically upon what infrared frequencies do to substances with a particular emphasis on how near infrared (NIR) spectroscopy is applied when it comes to determining some toxic substances like ochratoxin-A (OTA) in grape raisins. The study highlights the advantages of NIR, such as non-destructive analysis, rapid data acquisition, deep penetration, and sensitivity to organic compounds. Spectral data was collected using the Texas Instruments DLP NIRscan Nano Evaluation Module Spectrometer and used for image classification with convolutional neural networks (CNNs). The layers of CNNs are discussed and information about what they do and how they work is given. The importance of activation functions and classification algorithms is also emphasized, showing that NIR spectroscopy and deep learning techniques are potentially effective tools in food safety and quality control. In quality control applications in the food industry, this research underscores the significance of spectroscopy and AI-based methods.

Literature Review: Research in the literature shows that chemical and analytical methods are generally used for ochratoxin detection in dried grapes. However, these methods include some challenges and limitations. For example, these methods are often time-consuming, cause irreversible damage to samples, costly and complex. It has been observed that AI-based systems have yet to be applied innovatively in this field. The integration of AI technologies could enable the development of a faster, more sensitive, and reliable method for ochratoxin detection. By analyzing large data sets, AI can achieve more precise results and automate the detection process. Therefore, further study and improvement of AI-based techniques may represent an important advance in lowering chances in the food trade. By employing these advances, better surveillance, and management of possible contaminants such as ochratoxin in raisins can potentially be achieved.

Desing: During the design phase, data were first collected in the 900-1700 nm range using a DLP NIR SCAN NANO spectrometer. This was an important step to determine the spectral characteristics of contaminated and healthy raisins. The spectrometer collected data by contacting the grapes through plastic bags and stored using Python to analyze this data.

During the data collection process, the effect of different environments was also tested, and optimal results were obtained in a black sealed box, which prevents light from entering from the outside.

In the data analysis and visualization phase, statistical analysis of the collected spectral data was performed and visualized with graphs. These analyses revealed that there were marked differences between contaminated and healthy grapes. Significant changes were observed at wavelengths between 1400 nm and ~1670 nm.

During the model training and testing process, a high accuracy model was developed using pre-trained classification algorithms and a scratch CNN model. The model was split into 500 training and 200 test data sets and optimized by experimenting with different features (e.g. data augmentation). The results showed that the custom CNN model had 99% training accuracy and 85% testing accuracy. Finally, the model was integrated into the Raspberry Pi and its usability was tested under field conditions. This step was important for demonstrating the effectiveness of the designed model in real-world applications.

Merhods: In the Methods section, spectral analysis and machine learning classification of contaminated and healthy grapes were performed. First, contaminated, and healthy grapes were packaged separately in a sterile environment. Hygiene rules were strictly followed, and the risk of contamination was minimized. Spectral data were collected on the packed grapes using a DLP NIR SCAN NANO spectrometer. The optimal data collection environment was determined to be in a black sealed box. The collected spectral data was then analyzed using Python. According to the analysis results, the average absorbance value of contaminated grapes was 0.10AU lower and the average reflectance value was 0.018AU higher than healthy grapes. The stability and distribution of the obtained data were analyzed graphically, and reliable data were collected. Finally, the prepared images were divided into training and test datasets and model training was performed using pre-trained and customized classification algorithms.

Results and Discussion: Various models were experimented on in this study, among them include different pretrained models (VGG, LSTM, Resnet50) as well as a custom CNN model which was developed from the ground up. From the experiments carried out, it was found out that the CNN model was the best performing model and yielded the best results. Different

optimizers, hyperparameter settings and layers were tried to improve the accuracy and consistency of the model.

The following results were obtained in different experiments:

For VGG16, the accuracy ranged from 85.5% to 87% with different settings.

For LSTM, the accuracy ranged from 68.1% to 90% with different settings.

For the CNN model, training was performed with images and different trials with various settings. The best results were achieved, with a training accuracy of up to 97% and a test accuracy of up to 85.5%.

The overall flow of the experiments shows that accuracy generally shows a steady increase, and the loss generally shows a steady decrease. However, although some fluctuations were observed, it was generally a stable training process.

The results of the study show that the combination of NIR Spectrometry and artificial intelligence is effective in detecting OTA contamination in raisins. These methods have the potential for wide application in agriculture and food safety. Future research could consider the development of portable and mobile versions for ochratoxin detection in other agricultural products. In this way, safer and higher quality agricultural products can be obtained, contributing to human health.

REFERENCES

[1] S. ALBAWI and T. A. MOHAMMED, "Understanding of a Convolutional Neural Network," *nternational Conference on Engineering and Technology (ICET),* 2017.

[2] N. Shahadat, "Convolutional Layer Reduction from Deep Convolutional Networks," *26th International Conference on Computer and Information Technology (ICCIT),* 2023.

[3] S. LOFFE and C. SZEGEDY, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *Proceedings of the 32nd International Conference on Machine Learning,* no. 38, pp. 448-456, 2015.

[4] N. SRİVASTAVA, G. HİNTON, A. KRİZHEVSKY, I. SUTSKEVER and R. SALAKHUTDİNOV, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research,* no. 15, pp. 1929-1958, 6 2014.

[5] A. "Convolutional Neural Network. In this article, we will see what are… | by Arc," 25 December 2018. [Online]. Available: https://towardsdatascience.com/convolutional-neural-network-17fb77e76c05. [Accessed 12 May 2024].

[6] S. KILIÇARSLAN, K. ADEM and M. ÇELİK, "An Overview of the Activation Functions Used in Deep Learning Algorithms," *Journal of New Results in Science,* vol. 10, no. 3, pp. 75-88, 2021.

[7] A. S. ÜNAL and P. D. B. ALPERTUNGA, "KURU ÜZÜM VE ÜRÜNLERİNDE OKRATOKSİN A VARLIĞININ ARAŞTIRILMASI," 2009.

[8] G. TÜRKÖZ BAKIRCI, F. ÇAKMAK and D. ÖZDEMİR, "Ege Bölgesi'nde Satışa Sunulan Kuru Üzümlerde Okratoksin A ve Küf İlişkisi," *Sidas Medya Akademik Gıda,* vol. 14, no. 4, pp. 407-411, 2016.

[9] Z. ASLANOĞLU, "KURU ÜZÜMDE OKRATOKSİN A VE FUMONİSİN B2 VARLIĞININ İNCELENMESİ," p. 19, 2014.

[10] A. MEDINA, M. V.-A. FRANCİSCO, J. V. Gemino-Adelantado, R. Mateo, F. Mateo and M. Jiménez, "New method for determination of ochratoxin A in beer using zinc acetate and solid-phase extraction silica cartridges," *Journal of Chromatography,* vol. 1121, pp. 178-183, 2006.

[11] C. Frenette, R. J. Paugh, M. Tozlovanu, M. Juzio, A. Pfohl-Leszkowicz and R. A. Manderville, "Structure-activity relationships for the fluorescence of ochratoxin A: insight for detection of ochratoxin A metabolites," *Analytica Chimica Acta,* vol. 6, no. 17, pp. 153-161, 2008.

[12] E. ONAN and H. ÇOBAN, "ÜZÜM VE ŞARAPTA OLASI BİR TEHLİKE: OKRATOKSİN A," *Ziraat Fakültesi Dergisi,* vol. 20, no. 39, pp. 53-57, 2006.

[13] H. ŞAHİN, "Çekirdeksiz kuru üzüm ihracatı 2021-2022 sezonunda 250 bin tonu aştı," 18 September 2022. [Online]. Available: https://www.aa.com.tr/tr/ekonomi/cekirdeksiz-kuru-uzum-ihracati-2021-2022-sezonunda-250-bin-tonu-asti/2687991. [Accessed 6 May 2024].

[14] P. R. Griffiths and J. A. Haseth, Fourier Transform Infrared Spectrometry, Wiley-Interscience, 2007.

[15] B. G. Obsborne and T. Fearn, Near Infrared Spectroscopy in Food Analysis, Longman Scientific & Technical, 1986.

[16] J. Workman and L. Weyer, Practical Guide and Spectral Atlas for Interpretive Near-Infrared Spectroscopy, CRC Press, 2012.

APPENDICES

Appendix A

DATA ANALYSIS CODE;

```
import os
import pandas as pd

# Veri klasörlerini belirt
okra_klasoru = "F:\_700_CSV_Data\Okralı"
normal_klasoru = "F:\_700_CSV_Data\Sağlıklı"

# Okra verilerini depolamak için boş bir DataFrame oluştur
okra_veriler = pd.DataFrame()

# Okra klasöründeki .csv dosyalarını oku
for dosya in os.listdir(okra_klasoru):
   if dosya.endswith('_r.csv'):
     dosya_yolu = os.path.join(okra_klasoru, dosya)
     veri = pd.read_csv(dosya_yolu)
     okra_veriler = pd.concat([okra_veriler, veri], ignore_index=True)

# Normal verilerini depolamak için boş bir DataFrame oluştur
normal_veriler = pd.DataFrame()

# Normal klasöründeki .csv dosyalarını oku
for dosya in os.listdir(normal_klasoru):
  if dosya.endswith('_r.csv'):
    dosya_yolu = os.path.join(normal_klasoru, dosya)
    veri = pd.read_csv(dosya_yolu)
    normal_veriler = pd.concat([normal_veriler, veri], ignore_index=True)

# Okra verilerinin istatistiksel özetini görüntüle
print("Statistical Summary of OTA Dataset:")
print(okra_veriler.describe())

# Normal verilerinin istatistiksel özetini görüntüle
print("Statistical Summary of Healthy Dataset:")
print(normal_veriler.describe())
```

Appendix B

DATA VISUALIZATION CODE;

```
import os
import pandas as pd
import matplotlib.pyplot as plt

# Veri klasörlerini belirt
okrali_klasoru = "F:\_700_CSV_Data\Okralı"
normal_klasoru = "F:\_700_CSV_Data\Sağlıklı"

def cizdir_ve_kaydet(klasor, title, dosya_adi):
    plt.figure(figsize=(10, 6))
    dosyalar = os.listdir(klasor)
    for dosya in dosyalar:
        if dosya.endswith('_a.csv'):
            dosya_yolu = os.path.join(klasor, dosya)
            veri = pd.read_csv(dosya_yolu)
            plt.plot(veri['Wavelength (nm)'], veri['Absorbance (AU)'], label=f'{dosya[:-4]}')  # Her bir veri setini
farklı renkle çizdir
    plt.xlabel('Wavelength (nm)')
    plt.ylabel('Absorbance (AU)')
    plt.title(title)
    plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
    plt.grid(True)
   # plt.savefig(f'{dosya_adi}.png')  # Grafikleri png olarak kaydet
    plt.show()

# Okralı veriler için grafik çizdir ve kaydet
cizdir_ve_kaydet(okrali_klasoru, 'OTA Dataset', 'Reflectance_40_228_Okrali_700Data')

# Normal veriler için grafik çizdir ve kaydet
cizdir_ve_kaydet(normal_klasoru, 'Healthy Dataset', 'Reflectance_40_228_Normal_700Data')
```

Appendix C

CSV TO PNG CODE;

```
import os
import pandas as pd
import matplotlib.pyplot as plt

# CSV dosyalarının bulunduğu klasörlerin yolu
```

```
okrali_klasoru = r'C:\Users\AliBakiTURKOZ\OneDrive\Masaüstü\Engineering
Project\DATASETS\Main_Dataset\CSV\Birleştirilmiş\Okralı'
saglikli_klasoru = r'C:\Users\AliBakiTURKOZ\OneDrive\Masaüstü\Engineering
Project\DATASETS\Main_Dataset\CSV\Birleştirilmiş\Sağlıklı'

# Grafiklerin kaydedileceği klasörlerin adları
okrali_grafik_klasoru = r"C:\Users\AliBakiTURKOZ\OneDrive\Masaüstü\Engineering
Project\DATASETS\Main_Dataset\PNG\700_Sinirli_Cift\Okrali"
saglikli_grafik_klasoru = r"C:\Users\AliBakiTURKOZ\OneDrive\Masaüstü\Engineering
Project\DATASETS\Main_Dataset\PNG\700_Sinirli_Cift\Saglikli"

# Klasörleri oluşturma
if not os.path.exists(okrali_grafik_klasoru):
    os.makedirs(okrali_grafik_klasoru)

if not os.path.exists(saglikli_grafik_klasoru):
    os.makedirs(saglikli_grafik_klasoru)

# CSV dosyalarını işleme alma ve grafik çizme
def grafik_cek_kaydet(klasor_yolu, grafik_klasoru):
    # Klasördeki tüm dosyaları al
    dosyalar = os.listdir(klasor_yolu)

    # Her dosya için işlem yap
    for dosya_adi in dosyalar:
        # Dosya yolu
        dosya_yolu = os.path.join(klasor_yolu, dosya_adi)

        # Dosya uzantısı kontrolü
        if dosya_yolu.endswith('_a.csv'):
            # CSV dosyasını oku (Absorbans)
            veri_absorbans = pd.read_csv(dosya_yolu)

            # Corresponding reflectance file
            dosya_yolu_yansitma = dosya_yolu.replace('_a.csv', '_r.csv')
            # CSV dosyasını oku (Reflectance)
            veri_yansitma = pd.read_csv(dosya_yolu_yansitma)

            # Grafik çiz
            plt.plot(veri_absorbans['Wavelength (nm)'], veri_absorbans['Absorbance (AU)'], color='k',
label='Absorbance')
            plt.plot(veri_yansitma['Wavelength (nm)'], veri_yansitma['Reflectance (AU)'], color='r',
label='Reflectance')

        #  plt.legend()  # Etkinleştirme legendl
        #    plt.xlabel("Wavelength (nm)")  # X ekseni etiketi
        #    plt.ylabel("Value")  # Y ekseni etiketi
        #    plt.title(f"{dosya_adi} Veri Grafiği")  # Grafik başlığı
```

```
        plt.axis('off')  # Eksenleri kaldır
        plt.grid(False)

        # Eksen sınırlamalarını ayarlama
        plt.xlim(900, 1700)  # Wavelength için 900-1700 arası
        plt.ylim(0, 2.5)    # Absorbance için 0.5-2.5 arası

        # Grafik boyutlarını ayarla
        fig = plt.gcf()
        fig.set_size_inches(5.12, 5.12)  # İstenen boyutlara göre ayarla (kaç pixel istersek 100 de biri olarak yaz)

        # Grafikleri kaydet
        grafik_yolu = os.path.join(grafik_klasoru, dosya_adi.replace('.csv', '.png'))
        plt.savefig(grafik_yolu)

        # Grafikleri temizle
        plt.clf()
        plt.close()

# Okralı klasöründeki CSV dosyaları için grafikleri çiz ve kaydet
grafik_cek_kaydet(okrali_klasoru, okrali_grafik_klasoru)
# Sağlıklı klasöründeki CSV dosyaları için grafikleri çiz ve kaydet
grafik_cek_kaydet(saglikli_klasoru, saglikli_grafik_klasoru)

print("Grafikler başarıyla oluşturuldu ve kaydedildi.")
```

Appendix D

CNN CODE;

```
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from keras import layers, Sequential, models
import pathlib
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras import regularizers

from google.colab import drive
drive.mount('/content/drive')


"""### Set all the Constants"""
```

```
BATCH_SIZE = 64
IMAGE_SIZE = 512
CHANNELS=3
EPOCHS=400



"""### Import data into tensorflow dataset object"""

dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "/content/drive/MyDrive/AliBaki_TURKOZ_Engineering_Project/Data_Sets/Grafikler/Çift_Grafikler/700_Sinirsiz_Cift/500_Train",
    seed=12,
    shuffle=True,
    image_size=(IMAGE_SIZE,IMAGE_SIZE),
    batch_size=BATCH_SIZE
)

class_names = dataset.class_names
class_names

# Visualize Some of The Images From Dataset
plt.figure(figsize=(20, 20))
for image_batch, labels_batch in dataset.take(1):
    for i in range(12):
        ax = plt.subplot(3, 4, i + 1)
        plt.imshow(image_batch[i].numpy().astype("uint8"))
        plt.title(class_names[labels_batch[i]])
        plt.axis("off")

#Train, Test, Validation Split
def get_dataset_partitions_tf(ds, train_split=0.7, val_split=0.2, test_split=0.1, shuffle=True, shuffle_size=1000):
    assert (train_split + test_split + val_split) == 1

    ds_size = len(ds)

    if shuffle:
        ds = ds.shuffle(shuffle_size, seed=12)

    train_size = int(train_split * ds_size)
    val_size = int(val_split * ds_size)

    train_ds = ds.take(train_size)
    val_ds = ds.skip(train_size).take(val_size)
    test_ds = ds.skip(train_size).skip(val_size)

    return train_ds, val_ds, test_ds
```

```
train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset)

print("Size of Data is :{0} \nBatch size of Training Data is
:{1}\nBatch size of Validation Data is :{2} \nBatch size of Testing
Data is :{3} " .format(len(dataset), len(train_ds), len(val_ds),
len(test_ds)))

"""### Cache, Shuffle, and Prefetch the Dataset"""

train_ds =
train_ds.cache().shuffle(200).prefetch(buffer_size=tf.data.AUTOTUNE)
val_ds =
val_ds.cache().shuffle(100).prefetch(buffer_size=tf.data.AUTOTUNE)
test_ds =
test_ds.cache().shuffle(50).prefetch(buffer_size=tf.data.AUTOTUNE)

"""## Building the Model

### Creating a Layer for Resizing and Normalization
"""

resize_and_rescale = tf.keras.Sequential([
 layers.Resizing(IMAGE_SIZE, IMAGE_SIZE),
  layers.Rescaling(1./255),
])

"""### Data Augmentation
"""

data_augmentation = tf.keras.Sequential([
 #   layers.RandomFlip("horizontal_and_vertical"),
  #  layers.RandomRotation(0.2),
   # layers.RandomZoom(0.2),
     layers.RandomBrightness(0.2),
     #layers.GaussianNoise(0.1),
     #layers.RandomContrast(0.2),
])

"""#### Applying Data Augmentation to Train Dataset"""

train_ds = train_ds.map(
    lambda x, y: (data_augmentation(x, training=True), y)
).prefetch(buffer_size=tf.data.AUTOTUNE)

"""### Model Architecture
"""

input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
```

```
n_classes = 2

model = models.Sequential([
    resize_and_rescale,
    layers.Conv2D(32, kernel_size=(3, 3), activation='relu'),
#input_shape=input_shape),
  #  layers.Conv2D(32, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
   # layers.Dropout(0.3),  # Dropout for regularization
    layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
    #layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
   # layers.Dropout(0.3),
    layers.Conv2D(128, kernel_size=(3, 3), activation='relu'),
   # layers.Conv2D(128, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
   # layers.Dropout(0.3),
    layers.Conv2D(256, kernel_size=(3, 3), activation='relu'),
   # layers.Conv2D(256, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
   # layers.Dropout(0.3),
    layers.Conv2D(512, kernel_size=(3, 3), activation='relu'),
   # layers.Conv2D(512, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),
   # layers.Dense(1024, activation='relu',
kernel_regularizer=regularizers.l2(0.001)),
    #layers.Dropout(0.8),  # Dropout for regularization
    layers.Dense(512, activation='relu',
kernel_regularizer=regularizers.l2(0.001)),
    layers.Dropout(0.8),  # Dropout for regularization
    layers.Dense(n_classes, activation='sigmoid'),
])

model.build(input_shape=input_shape)

model.summary()

"""### Compiling the Model
"""

# Compile the model with a lower learning rate
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.00001),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=Fals
e),
    metrics=['accuracy']
)
```

```
#Decreasing Learning Rate
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(
   monitor='val_loss',
   factor=0.1,
    patience=7,
    verbose=1,              # Mesajları yazdır
    mode='auto',
    min_delta=0.0001,       # bu deger kadar degisim olmazsa ogrenme
oranini dusurur
    cooldown=0,
    min_lr=1e-7
)

# EarlyStopping callback'ini tanımlayın
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=15,
    verbose=1,              # Mesajları yazdır
    mode='auto',
    restore_best_weights=True
)
history = model.fit(
    train_ds,
    batch_size=BATCH_SIZE,
    validation_data=val_ds,
    verbose=1,
    epochs=EPOCHS,
    callbacks=[early_stopping, reduce_lr]
)

scores = model.evaluate(test_ds)

"""Scores is just a list containing loss and accuracy value

### Plotting the Accuracy and Loss Curves
"""

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

import matplotlib.pyplot as plt

# Doğruluk grafiği
plt.subplot(1, 2, 1)
plt.plot(acc, label='Eğitim Doğruluğu')
```

```
plt.plot(val_acc, label='Doğrulama Doğruluğu')
plt.title('Eğitim ve Doğrulama Doğruluğu')
plt.xlabel('Epoch')
plt.ylabel('Doğruluk')
plt.legend()

# Kayıp grafiği
plt.subplot(1, 2, 2)
plt.plot(loss, label='Eğitim Kaybı')
plt.plot(val_loss, label='Doğrulama Kaybı')
plt.title('Eğitim ve Doğrulama Kaybı')
plt.xlabel('Epoch')
plt.ylabel('Kayıp')
plt.legend()

plt.show()

"""### Run prediction on a sample image"""


import numpy as np
for images_batch, labels_batch in test_ds.take(1):


    first_image = images_batch[0].numpy().astype('uint8')
    first_label = labels_batch[0].numpy()


    print("first image to predict")
    plt.imshow(first_image)
    print("actual label:",class_names[first_label])


    batch_prediction = model.predict(images_batch)
    print("predicted
label:",class_names[np.argmax(batch_prediction[0])])


"""### Write a function for inference"""


def predict(model, img):
    img_array =
tf.keras.preprocessing.image.img_to_array(images[i].numpy())
    img_array = tf.expand_dims(img_array, 0)


    predictions = model.predict(img_array)
```

```
    predicted_class = class_names[np.argmax(predictions[0])]
    confidence = round(100 * (np.max(predictions[0])), 2)
    return predicted_class, confidence


"""**Now run inference on few sample images**"""


plt.figure(figsize=(15, 20))
for images, labels in test_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))


        predicted_class, confidence = predict(model, images[i].numpy())
        actual_class = class_names[labels[i]]


        plt.title(f"Actual: {actual_class},\n Predicted:
{predicted_class}.\n Confidence: {confidence}%")


        plt.axis("off")

# Model Save

from tensorflow.keras.models import load_model
model.save("/content/drive/MyDrive/AliBaki_TURKOZ_Engineering_Project/s
aved_model/Deneme_Çift_Sinirli_CNN_Model_20_.h5")
```

Appendix E

VGG16 MODEL

```
import os
import numpy as np
from keras.preprocessing.image import load_img, img_to_array
from keras.applications.vgg16 import VGG16, preprocess_input as preprocess_vgg16
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from keras.models import Sequential
from keras.layers import Flatten, Dense
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from scipy.ndimage import gaussian_filter
from keras.callbacks import EarlyStopping
from keras.layers import Dropout
from keras.regularizers import l2
```

```
# Google Drive API
from google.colab import drive

# Drive bağlantısını oluştur
drive.mount('/content/gdrive')

# Veri yolları
normal_klasoru =
"/content/gdrive/MyDrive/AliBaki_TURKOZ_Engineering_Project/Data_Sets/Grafikler/Saglikli_Grafikler_512x512"
okra_klasoru =
"/content/gdrive/MyDrive/AliBaki_TURKOZ_Engineering_Project/Data_Sets/Grafikler/Okrali_Grafikler_512x512"

# Veri ve etiketlerin depolanacağı boş listeleri oluştur
X = []
y = []

# Fonksiyon: Görüntü yükleme ve bulanıklık ekleme
def load_and_blur_image(image_path, target_size=(256, 256)):
    img = load_img(image_path, target_size=target_size)
    img_array = img_to_array(img)
    blurred_img_array = gaussian_filter(img_array, sigma=1.5)  # Bulanıklık ekleme
    return blurred_img_array

# Zararlı sınıf için verileri yükleme ve bulanıklık ekleme
for dosya_adi in os.listdir(okra_klasoru):
    dosya_yolu = os.path.join(okra_klasoru, dosya_adi)
    img = load_and_blur_image(dosya_yolu)
    img = preprocess_vgg16(img)
    X.append(img)
    y.append(0)  # Zararlı sınıf

# Normal sınıf için verileri yükleme ve bulanıklık ekleme
for dosya_adi in os.listdir(normal_klasoru):
    dosya_yolu = os.path.join(normal_klasoru, dosya_adi)
    img = load_and_blur_image(dosya_yolu)
    img = preprocess_vgg16(img)
    X.append(img)
    y.append(1)  # Normal sınıf



# Veriyi Numpy dizilerine dönüştürme
X = np.array(X)
y = np.array(y)

# Verileri karıştırma
X, y = shuffle(X, y)

# Eğitim ve test verisi olarak ayırma
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Modeli yükleme
vgg16_model = VGG16(weights='imagenet', include_top=False, input_shape=(256, 256, 3))

# Modeli eğitilemez hale getirme
vgg16_model.trainable = True
```

```
# Tam bağlı katmanları oluşturma
model = Sequential([
 vgg16_model,
 Flatten(),
 Dense(256, activation='relu'),
 Dropout(rate=0.5),  # Dropout oranını artırdık
 Dense(128, activation='relu'),
 Dropout(rate=0.5),  # Dropout oranını artırdık
 Dense(1, activation='sigmoid')
])

# Modeli derleme
optimizer = Adam(learning_rate=0.0001)  # Öğrenme oranını düşürme
model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])

# EarlyStopping callback'i oluşturun
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Modeli eğitme
history = model.fit(X_train, y_train, batch_size=32, epochs=40, validation_split=0.2, callbacks=[early_stopping])

# Modeli test verisiyle değerlendirme
loss, accuracy = model.evaluate(X_test, y_test)
print(f'loss: {loss}')
print(f'VGG16 Modeli Test Doğruluk Oranı: {accuracy * 100:.2f}%')
```

Appendix F

LSTM MODEL CODE

```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam

# Google Drive'ı bağlayın
from google.colab import drive
drive.mount('/content/drive')

# Veri klasörlerinin yolu
normal_klasoru =
"/content/drive/MyDrive/AliBaki_TURKOZ_Engineering_Project/Data_Sets/Gr
afikler/Manually_Splitted_Datas/400Data_Train/Sağlıklı_Çift_512x512"
```

```
okra_klasoru =
"/content/drive/MyDrive/AliBaki_TURKOZ_Engineering_Project/Data_Sets/Gr
afikler/Manually_Splitted_Datas/400Data_Train/Okralı_Çift_512x512"

# Okra ve normal verilerini saklamak için boş liste oluştur
okra_gorselleri = []
normal_gorselleri = []

# Okra dosyalarını oku ve gorselleri okra_gorselleri listesine ekle
for dosya in os.listdir(okra_klasoru):
    if dosya.endswith('.png'):
        dosya_yolu = os.path.join(okra_klasoru, dosya)
        gorsel = load_img(dosya_yolu, color_mode='grayscale',
target_size=(512,512))
        gorsel = img_to_array(gorsel)
        okra_gorselleri.append(gorsel)

# Normal dosyaları oku ve gorselleri normal_gorselleri listesine ekle
for dosya in os.listdir(normal_klasoru):
    if dosya.endswith('.png'):
        dosya_yolu = os.path.join(normal_klasoru, dosya)
        gorsel = load_img(dosya_yolu, color_mode='grayscale',
target_size=(512,512))
        gorsel = img_to_array(gorsel)
        normal_gorselleri.append(gorsel)

# Okra ve normal gorsellerini birleştir
okra_gorselleri = np.array(okra_gorselleri)
normal_gorselleri = np.array(normal_gorselleri)
X = np.concatenate((okra_gorselleri, normal_gorselleri))

# Etiketleri oluştur
okra_etiketleri = np.ones(len(okra_gorselleri))
normal_etiketleri = np.zeros(len(normal_gorselleri))
y = np.concatenate((okra_etiketleri, normal_etiketleri))

# Verileri standartlaştırma
scaler = StandardScaler()
X = scaler.fit_transform(X.reshape(X.shape[0], -1))
X = X.reshape(X.shape[0], 1, 512*512)  # LSTM giriş şekline uyacak
şekilde yeniden şekillendir

# Verileri eğitim ve test setlerine ayırma
X_egitim, X_test, y_egitim, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

from sklearn.model_selection import KFold

# K-fold cross validation için k değerini belirleyin
```

```
k = 5
kf = KFold(n_splits=k, shuffle=True, random_state=42)

# Modelinizi her bir fold için eğitin ve değerlendirin
for i, (train_index, val_index) in enumerate(kf.split(X)):
    print(f"Fold {i+1}/{k}")
    X_train, X_val = X[train_index], X[val_index]
    y_train, y_val = y[train_index], y[val_index]

    # Modeli yeniden oluşturun (her fold'da sıfırlamak önemlidir)
    model = Sequential([
        LSTM(128, input_shape=(1, 512*512), return_sequences=True),
        Dropout(0.5),
        LSTM(64, return_sequences=False),
        Dropout(0.5),
        Dense(64, activation='relu'),
        Dropout(0.5),
        Dense(1, activation='sigmoid')
    ])

    # Özel bir learning rate ile Adam optimizer oluşturma
    optimizer = Adam(learning_rate=0.00005)

    # Modeli derleyin
    model.compile(optimizer=optimizer, loss='binary_crossentropy',
metrics=['accuracy'])

    # Early stopping tanımlama
    early_stopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)

    # Modeli eğitme ve early stopping kullanma
    model.fit(X_train, y_train, epochs=40, batch_size=16,
validation_data=(X_val, y_val), callbacks=[early_stopping])

# Sonuçları ortalama olarak değerlendirin
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test setinde kayıp: {loss}")
print(f"Test setinde doğruluk: {accuracy}")
```

Appendix G

TESTİNG CODE;

```
import os
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, log_loss
from keras.preprocessing.image import load_img, img_to_array
from keras.utils import to_categorical
from tensorflow.keras.models import load_model

# Okra ve normal klasörlerinin yolları
okra_klasoru =
"/content/drive/MyDrive/AliBaki_TURKOZ_Engineering_Project/Data_Sets/Gr
afikler/Çift_Grafikler/700_Sinirli_Cift/200_Test/Okrali"
normal_klasoru =
"/content/drive/MyDrive/AliBaki_TURKOZ_Engineering_Project/Data_Sets/Gr
afikler/Çift_Grafikler/700_Sinirli_Cift/200_Test/Saglikli"

# Verileri ve etiketleri depolamak için boş listeler
X = []
y = []

# Okra verilerini yükleme ve etiketleme
for dosya_adi in os.listdir(okra_klasoru):
    dosya_yolu = os.path.join(okra_klasoru, dosya_adi)
    img = load_img(dosya_yolu, target_size=(512, 512))
    img_array = img_to_array(img)
    X.append(img_array)
    y.append(0)  # Okra sınıfı

# Normal verilerini yükleme ve etiketleme
for dosya_adi in os.listdir(normal_klasoru):
    dosya_yolu = os.path.join(normal_klasoru, dosya_adi)
    img = load_img(dosya_yolu, target_size=(512, 512))
    img_array = img_to_array(img)
    X.append(img_array)
    y.append(1)  # Normal sınıfı

# Veriyi ve etiketleri Numpy dizilerine dönüştürme
X = np.array(X)
y = np.array(y)

# Verileri karıştırma
sirali_indeksler = np.arange(X.shape[0])
np.random.shuffle(sirali_indeksler)
```

```
X = X[sirali_indeksler]
y = y[sirali_indeksler]

# Modeli yükleyip verileri tahmin etme
model =
load_model('/content/drive/MyDrive/AliBaki_TURKOZ_Engineering_Project/s
aved_model/Best Models/Deneme_Çift_CNN_Model_7_%84.5_.h5')  #
Kaydedilmiş modeli yükleme
predictions = model.predict(X)  # Verileri tahmin etme

# Gerçek etiketlerin ve tahminlerin hazırlanması
true_labels = y
true_labels_cat = to_categorical(true_labels, num_classes=2)  # Gerçek
etiketlerin kategorik hale getirilmesi
pred_prob_cat = predictions  # Tahmin olasılıklarının hesaplanması

# Doğruluk oranını ve kaybı hesaplama
accuracy = accuracy_score(true_labels, np.argmax(predictions, axis=1))
loss = log_loss(true_labels_cat, pred_prob_cat)

print(f"Doğruluk Oranı: {accuracy}")
print(f"Kayıp: {loss}")

# Her aşamadaki tahmin ve gerçek etiketleri yazdırma
print("Tahminler ve Gerçek Etiketler:")
for i in range(len(true_labels)):
    tahmin = 'Okratoksinli' if np.argmax(predictions[i]) == 0 else
'Sağlıklı'
    gerçek_etiket = 'Okratoksinli' if true_labels[i] == 0 else
'Sağlıklı'
    print(f"Tahmin: {tahmin} - Gerçek Etiket: {gerçek_etiket}")

# Accuracy oranlarının yazdırılması
accuracy_list = [accuracy_score(true_labels[:i+1],
np.argmax(predictions[:i+1], axis=1)) for i in range(len(true_labels))]
print("Her Aşamadaki Accuracy Oranları:")
for i, acc in enumerate(accuracy_list):
    print(f"Veri sayısı: {i+1}, Accuracy: {acc}")

# Loss ve accuracy grafiğinin çizdirilmesi
loss_list = []
mean_loss_list = []
for i in range(len(true_labels)):
    loss_list.append(log_loss(true_labels_cat[:i+1],
pred_prob_cat[:i+1]))
    mean_loss_list.append(np.mean(loss_list))  # Tüm epochlar için loss
ortalaması alınıyor

# Accuracy'lerin ortalamasını alalım
```

```
mean_accuracy_list = [np.mean(accuracy_list[:i+1]) for i in
range(len(accuracy_list))]

plt.figure(figsize=(15, 5))
plt.subplot(1, 3, 1)
plt.plot(range(len(true_labels)), accuracy_list, marker='o',
linestyle='-', label='Accuracy')
plt.title('Accuracy')
plt.xlabel('Veri Sayısı')
plt.ylabel('Accuracy')
plt.grid(True)
plt.legend()

plt.subplot(1, 3, 2)
plt.plot(range(len(true_labels)), mean_loss_list, marker='o',
linestyle='-', label='Mean Loss')
plt.title('Mean Loss')
plt.xlabel('Veri Sayısı')
plt.ylabel('Loss')
plt.grid(True)
plt.legend()

plt.subplot(1, 3, 3)
plt.plot(range(len(true_labels)), mean_accuracy_list, marker='o',
linestyle='-', color='orange', label='Mean Accuracy')
plt.title('Mean Accuracy')
plt.xlabel('Veri Sayısı')
plt.ylabel('Accuracy')
plt.grid(True)
plt.legend()

plt.tight_layout()
plt.show()
```