Almond Variety Detection using Deep Learning

Benarous Ahmed Omar Farouq - a41369

Thesis presented to the School of Technology and Management in the scope of the Master in Information Systems.

Supervisors:

Prof. Maria João Tinoco Varanda Pereirar

This document does not include the suggestions made by the board.

Bragança

2019-2020

Dedication

I dedicate my dissertation work to my family . A special feeling of gratitude to my loving parents, whose words of encouragement and push for tenacity ring in my ears. I also dedicate this dissertation to my many friends who have supported me throughout the process. I will always appreciate all they have done.

Acknowledgment

In the name of Allah, the Most Gracious, the Most Merciful

I would like to express my deep sense of gratitude, respect and heartfelt thanks to Prof. Maria João Tinoco Varanda Pereira, for her great contributions, encouragement and support during the thesis writing, understanding and patience that have accompanied me during the research time.

I would like to acknowledge and thank to ESTIG-IPB school division for providing me of the necessary materials needed to conduct my research and providing any assistance requested.

I would like to acknowledge and thank to ESA-IPB school division for allowing me to conduct my research and providing any assistance requested. Special thanks goes to the members of the lab Professors Nuno Rodrigues and José Alberto Pereira for their continued support.

Finally I would like to thank all professor of the hassiba benbouali university where i have started my journey of studying computer science. And special thank and acknowledgment to all community of computer science , this dynamic and healthy community is the reason why computer science, which is very new

field compared to other sciences, has evolved so dramatically fast and now have huge impact on almost everything.

Abstract

Quality is the major factor for modern industries because the high-quality of products is the basis for success in today's highly competitive market, so improving the product quality is not a choice is paramount to any business to be even competitive.

This thesis aims to solve one of the many problems that exist in today's market, with regard to the quality and how to evaluate the quality before putting the product in the market.
We have implemented a Deep-learning based technique that helps classify/identify almonds based only on their visual features, this can help in many ways from which we can mention: supply chain optimization, Sorting food, and matching customer taste ....

In the conclusion of this thesis, we propose many features that can be developed that can be added to our model, from scaling it to other products, to deploying it, in order to reach the full potential of our solution.

Keywords: Deep learning, Computer Vision, Food Quality

Resumo

A qualidade é o principal fator para as indústrias modernas porque a alta qualidade dos produtos é a base para o sucesso no mercado altamente competitivo de hoje, portanto, melhorar a qualidade do produto não é uma escolha é fundamental para qualquer empresa ser ainda competitiva.

Esta tese visa solucionar um dos muitos problemas que existem no mercado atual, no que diz respeito à qualidade e como avaliar a qualidade antes de colocar o produto no mercado.
Implementamos uma técnica baseada em aprendizagem profunda que ajuda a classificar / identificar amêndoas com base apenas em seus recursos visuais, isso pode ajudar de várias maneiras, das quais podemos mencionar: otimização da cadeia de fornecimento, classificação automática de alimentos e resposta personalizada/adaptada ao cliente.

Na conclusão desta tese, propomos muitos recursos que podem ser desenvolvidos e que podem ser adicionados ao nosso modelo, desde o dimensionamento para outros produtos, até a implantação, a fim de atingir todo o potencial da nossa solução.

Palavras-chave: Deep learning, Computer Vision, Food Quality .

Contents

List of Figures

Acronyms

API Application Programming Interface.

BE Back End.

CV Computer vision.

DAO Data Access Object.

DB Data Base.

DL Deep learning.

ESTiG Escola Superior de Tecnologia e Gestão.

FE Front End.

HTTP HyperText Transfer Protocol.

IPB Instituto Politécnico de Bragança.

LSTM Long-Short term memory.

ML Machine learning.

NN Neural Network.

Chapter 1

Introduction

This thesis intends to find a solution for a problem that CIMO (Mountain Research Center) of Polytechnic Institute of Bragança faces when dealing with a big variety of almonds. In this first chapter, we will be first introducing the general view of the problem that we are trying to solve and also introducing the field of deep learning and how it can be used to solve similar problems and second deploying the solution as SAS (software as service) allowing future students to scale/build on top of it to solve similar issues. This thesis is developed in CeDRI (Research Centre Digitization and Intelligent Robotics) of Polytechnic Institute of Bragança.

1.1 Introduction of the problem

Quality is a key factor for the modern food industry because the high-quality of products is the basis for success in today's highly competitive market. In the food industry, the quality evaluation still heavily depends on manual inspection, which is tedious, laborious, and costly, and is easily influenced by physiological factors, inducing subjective and inconsistent evaluation results. To satisfy the

increased awareness, sophistication, and greater expectation of consumers, it is necessary to improve the quality evaluation of food products [1]. If the quality evaluation is achieved automatically, production speed and efficiency can be improved in addition to the increased evaluation accuracy with an accompanying reduction in production costs.

Despite the fact that the food industry includes a diverse set of businesses such as agriculture, food processing, marketing, and sale, and that its value and importance in the increasingly globalized world is unsurpassed, it is surprising that the process of food quality evaluation is still mostly done manually by trained people. This approach is costly, inherently subjective, and prone to error. Thus, there exists a demand to increase the levels of objectivity, consistency, and efficiency in food quality evaluation. The digital image processing techniques and pattern recognition algorithms can be an important part of this endeavor [2]. Illumination«As a rapid, economic, consistent, and even more accurate and objective inspection tool, computer vision systems have been used increasingly in the food industry for quality evaluation purposes. The application potential of computer vision to the food industry has long been recognized. The food industry ranks among the top 10 industries using computer vision technology [3], which has been proven successful for the objective and non-destructive quality evaluation of several food products. Being an objective, rapid and non-contact quality evaluation tool, computer vision has been attracting much Research and development attention from the food industry, and rapid development has been increasingly taking place on quality inspection of a wide range of food products [2]. Combined with an illumination system, a computer vision system is typically based on a personal computer (PC) in connection with electrical and mechanical devices to replace human manipulative effort in the performance of a given process. Illumination is an important prerequisite

of image acquisition for food quality evaluation. The quality of the captured image can be greatly affected by the lighting condition. A high-quality image can help to reduce the time and complexity of the subsequent image processing steps, which can decrease the cost of an image processing system. A different application may require a different illumination strategy [4]. reported that most lighting arrangements could be grouped as one of the following: front lighting, backlighting, and structured lighting. For image processing algorithms, software implementation on a PC allows for rapid development, debug, and test. However, as image sizes grow larger and algorithms become more complex, the speed will be slower and cannot satisfy the requirement of high speed in real-time systems. Conversely, hardware implementation offers much greater speed than a software one. There are several viable options for hardware implementation of image processing algorithms, such as application-specific integrated circuits (ASICs), digital signal processors (DSPs), and field-programmable gate arrays (FPGAs). Although the speed can be improved by hardware implementation, one must consider the increase in development cost for creating a custom hardware design. Therefore, hardware designers usually use some sorts of PC programming environment to implement a design to verify functionality prior to a lengthy hardware design. In this thesis, we will focus on two main components first is the design and implementation of the deep learning model as well as the software part of it from developing the back-end Apis to the serving/deployment phase.

The learning technique is one of the essential features for food quality evaluation using computer vision, as the aim of computer vision is to ultimately replace the human visual decision-making process with automatic procedures. Computer vision tries to clone human behavior of performance in color, content, shape, and texture inspection [5] . Backed by powerful learning systems,

computer vision provides a mechanism in which the human thinking process is simulated artificially and can help humans in making complicated judgments accurately, quickly, and very consistently over a long period [3]. Learning techniques can be employed to learn meaningful or nontrivial relationships automatically in a set of training data and produce a generalization of these relationships that can be used to interpret new, unseen test data [1]. Therefore, using sample data, a learning system can generate an updated basis for improved classification of subsequent data from the same source, and express the new basis in intelligible symbolic form. Nevertheless, there is a definite need for research dealing with the combination of computer vision and learning techniques for food quality inspection [1].

1.2 Heads-Up of Deep Learning

The term Deep Learning was introduced to the machine learning community by Rina Dechter in 1986 and to artificial neural networks by Igor Aizenberg and colleagues in 2000, in the context of Boolean threshold neurons [3].

The first general, working learning algorithm for supervised, deep, feedforward, multilayer perceptrons was published by Alexey Ivakhnenko and Lapa in 1967. A 1971 paper described already a deep network with 8 layers trained by the group method of data handling algorithm [3], figure 1.1 visualizes the whole scope of the field.

Other deep learning working architectures, specifically those built for computer vision, began with the Neocognitron introduced by Kunihiko Fukushima in 1980 [34]. In 1989, Yann LeCun et al. applied the standard back-propagation algorithm, which had been around as the reverse mode of automatic differentiation since 1970 to a deep neural network with the purpose of recognizing

handwritten ZIP codes on mail. While the algorithm worked the training required 3 days.

In 2012, a team led by George E. Dahl won the "Merck Molecular Activity Challenge" using multi-task deep neural networks to predict the bio-molecular target of one drug In 2014, Hochreiter's group used deep learning to detect off-target and toxic effects of environmental chemicals in nutrients, household products, and drugs and won the "Tox21 Data Challenge" of NIH, FDA, and NCATS.

Significant additional impacts in image or object recognition were felt from 2011 to 2012. Although CNN's trained by back-propagation had been around for decades, and GPU implementations of NNs for years, including CNNs, fast implementations of CNNs with max-pooling on GPUs in the style of Ciresan and colleagues were needed to progress on computer vision. In 2011, this approach achieved for the first time superhuman performance in a visual pattern recognition contest. Also in 2011, it won the ICDAR Chinese handwriting contest, and in May 2012, it won the ISBI image segmentation contest. Until 2011, CNNs did not play a major role at computer vision conferences, but in June 2012, a paper by Ciresan et al. at the leading conference CVPR showed how max-pooling CNNs on GPU can dramatically improve many vision benchmark records. In October 2012, a similar system by Krizhevsky et al won the large-scale ImageNet competition by a significant margin over shallow machine learning methods. In November 2012, Ciresan et al.'s system also won the ICPR contest on analysis of large medical images for cancer detection, and in the following year also the MICCAI Grand Challenge on the same topic. In 2013 and 2014, the error rate on the ImageNet task using deep learning was further reduced, following a similar trend in large-scale speech recognition. The Wolfram Image Identification project publicized these improvements [6].

Image classification was then extended to the more challenging task of generating descriptions (captions) for images, often as a combination of CNNs and LSTMs. Some researchers assess that first Image-Net (October 2012) victory anchored the start of a "deep learning revolution" that has transformed the AI industry. In March 2019, Yoshua Bengio, Geoffrey Hinton, and Yann LeCun were awarded the Turing Award for conceptual and engineering breakthroughs that have made deep neural networks a critical component of computing.

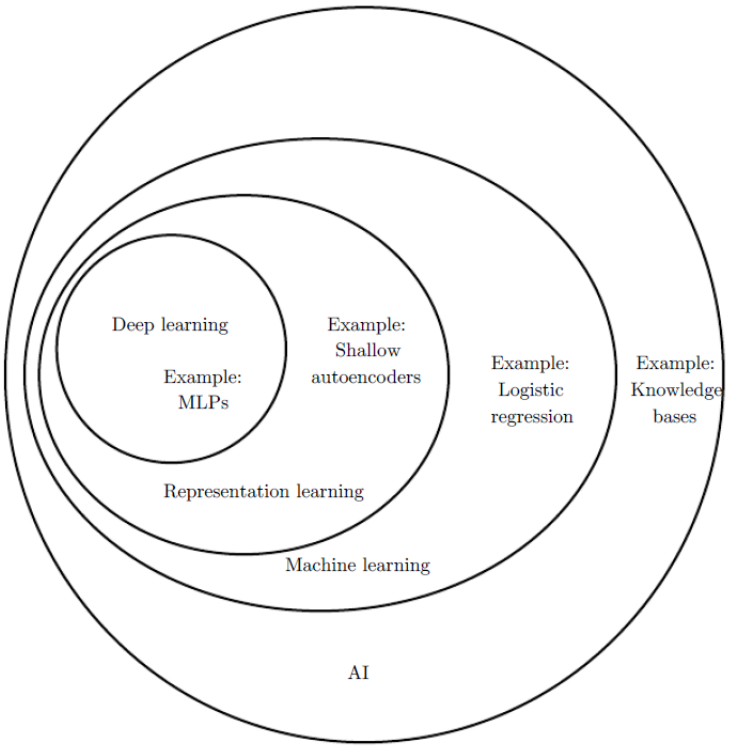


Figure 1.1: Visualisation of the whole scope of the field

1.3 Case of Study

Almonds are one of the most nutritious of all nuts. They have the highest protein content of any nut and are packed with vitamin E, calcium, iron, magnesium,

phosphorus, and zinc. As products with high nutrition are advantageous.

Almond production is about to double in Portugal with new plantations in the last decade, according to the National Center for Dried Fruits (CNCFS). According to official data, since 2010, this dried fruit has been growing and with the new plantations that will start to bear fruit in two or three years, the national production will double to 40 thousand tons. Plantations that are emerging in areas without cultivation tradition, such as the Alentejo. Still, both in this and other dried fruit productions the country still has plenty of room to grow and it is this perspective, as well as the technical and scientific progress that will be highlighted at the II National Symposium on Dried Fruits. The initiative brings together Portuguese, Spanish and Italian experts, and is open to technicians and farmers, as said Lusa Albino Bento of CNCFS, who organizes the symposium with the Portuguese Society of Agricultural Sciences (SCAP). Ten years ago, in Alentejo, there were "between 300 and 500 hectares of almond and now there are 10 thousand hectares" [7]. In Trás-os-Montes there was "a 10 percent growth" and, within two to three years, the country is expected to have "twice the production of almonds in a shell", equivalent to "40 thousand tons". The sub-row of dried fruits that still leads the way in Portugal is the chestnut, with production concentrated mainly in Trás-os-Montes. This is, according to Albino Bento, who is also a researcher at the Polytechnic Institute of Bragança, the only sub-row of dried fruits in which the country is a self-sufficient and net exporter. Direct production is worth about 70 million euros, a figure from which the almond is approaching with the prospect of increased production equivalent to a financial movement of about 60 million euros for the producer, according to those officials [7].

Still, Portugal is far from self-sufficient in almond production and continues to import, with the United States dominating 80 percent of the world market.

The Portuguese are also starting now to invest in walnut production, with an annual harvest of 4 thousand tons, and still shyly in hazelnut and pistachio or carob [7].

There are numerous factors such as product variety, different steps of planting, growing, and harvesting that play a role in promoting the quality of agricultural products. One of the most important post-harvest processing operations, directly related to improving the quality of the products, is the grading or sorting operation. Increasing the quality of the almond product utilizing a new and reliable technique is a key factor in exporting and economic profitability of the final product. For implementing such operations, both human vision (HV) and computer vision (CV) are being used. However, the human-based vision methods are becoming less attractive due to their high costs, low speeds, requiring experienced staff for grading of the product, and low accuracies. In recent years, the application of advanced techniques based on Computer Vision for grading different agricultural products due to its high accuracy, low cost, and high speed has become more widespread, the software solution that we will propose in this thesis would be first dedicated to the Escola Superior Agrária de Bragança labs and after the proof of concept it shall be deployed to serve and help almond Producers in the Region [7].

Chapter 2

State of the Art and background

In this chapter we will be discussing the state of the art related to our case of study, we have chosen two works to review where both of them aim to classify the almonds using different feature extraction techniques.

2.1 Introduction

Learning techniques have been applied increasingly for food quality using computer vision, which includes an artificial neural network, statistical learning, fuzzy logic, genetic algorithm, and decision tree. Artificial neural networks (ANN) and statistical learning (SL) remain the primary learning methods in the field of computer vision for food quality evaluation. Among the applications of learning algorithms in computer vision for food quality evaluation, most of them are for classification and prediction, however, there are also some for image segmentation and feature selection, In this Section, we will be discussing some of the methods used to try to solve similar problems to the one we are tackling.

2.2 Review/Comparison between different techniques

Autistic Based Classification

The first study that I have encountered while working on this subject was one by Simin Khalesi , Asghar Mahmoudi and Hosainpour Adel entitled " Detection of Walnut Varieties Using Impact Acoustics and Artificial Neural Networks (ANNs) " [8]. In this paper, they have built an acoustic-based intelligent system that classifies different types of walnut by extracting their autistic features. The system consisted of a feeding platform, an impact plate, an acoustic unit, and a PC based data acquisition system. A 2-meter V-shaped steel profile was used as a feeding platform and a chain was used for adjusting the slid angle. At 30o inclination of the slid, the best trajectory for walnuts was obtained. Flatter angles of incline would tend to bounce twice before falling off of the plate since in a more inclined angle the trajectory would not be as consistent. This angle was determined by trial and error. Preliminary experimental results indicated that steel plates proved better than glass or wooden for separating different walnut genotypes sounds. The impact plate was made up of a polished block of stainless steel approximately 150×150×20 mm. The mass of a single walnut is negligible compared to the mass of the impact plate, hence the possibility of vibrations from the plate interfering with acoustic emissions from nuts was minimized. The drop distance from the endpoint of the feeder to the impact plate was 25 cm. The impact plate in a horizontal situation produces proper sound signals for complementary processing. A low-cost Panasonic Electret capsule microphone (VM-034CY model), sensitive to frequencies up to 100 kHz, was used for capturing impact sound signals. The microphone was installed

inside an isolated acoustic chamber to eliminate environmental noise effects. To prevent the chamber from acoustic reflections, it was filled with glass wool. Microphone output was sent to a PC based data acquisition system, where it was digitized using a sound card (Intel® 82801 BA/BAM AC‘97Audio controller) at a sampling frequency of 44.1 kHz, with 16 bit resolution. The personal computer was used for acquiring, saving and processing of data as well as classification samples. A schematic diagram describing the overall detection system is shown in figure 2.1 and figure 2.2. Parallel hardware and software architecture is used to perform the described duty [8].

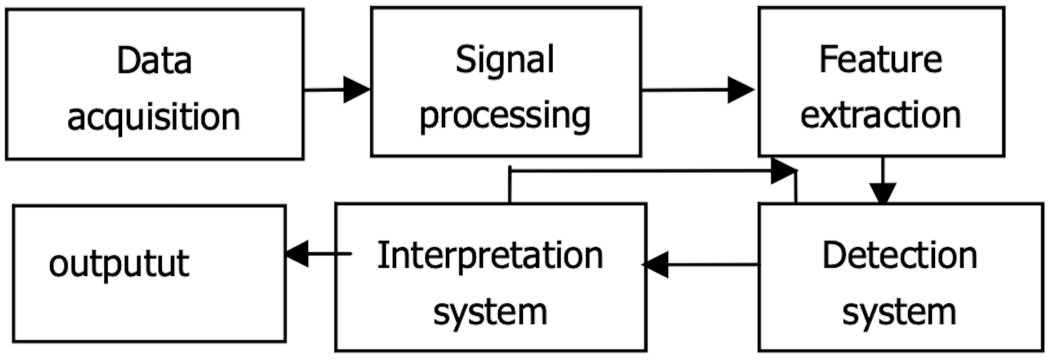


Figure 2.1: The block Diagram of the detection system [8] .

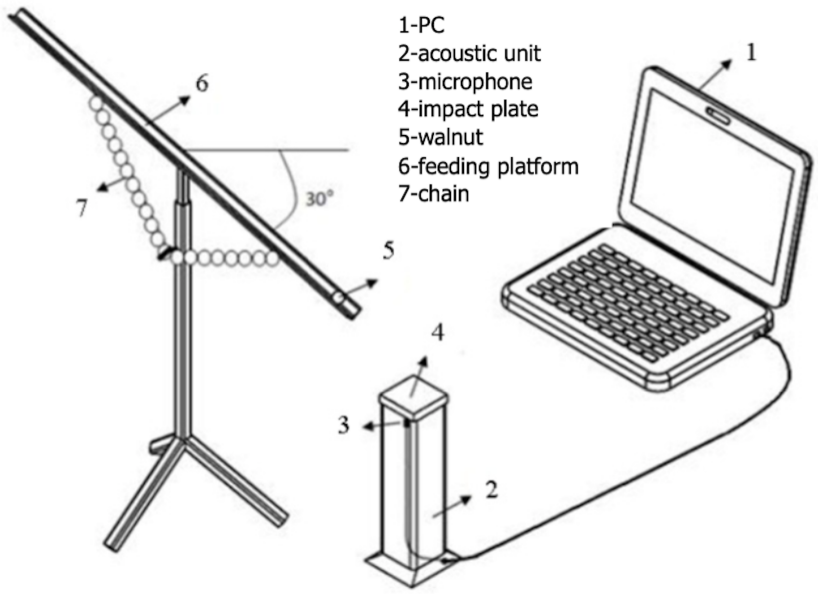


Figure 2.2: The schematic diagram of the experimented apparatus [8].

To find the best combinations of potential features and optimal ANN configuration, they have used 15 different combinations of principal component features that were selected and tested by a neural network (Figure 2.3). These features were fed to the ANN models and their performances were determined by evaluation of the mean square error (MSE), correct detection rate (CDR), and correlation coefficient (r). In summary, the best combination was 16 amplitudes and 31 PSD features. The phase angles features had not enough potential to discriminate between sangi and kaghazi walnuts. The final structure of the network was 47 input nodes, 18 hidden nodes, and 2 output nodes. This relatively low number of input vector was used to guard against the possibility of over-fitting the neural network [8].

Features and their eliminated components variance	Number of selected features	Correct Detection Rate (CDR) %		correlation coefficient (r)		Mean Square Error (MSE)	
		Kagha zi	Sangi	Kagha zi	Sangi	Kagha zi	Sangi
Amp0.001+Psd0.001+Ang0.001	22+3+61	93.77	97.43	0.93	0.93	0.0374	0.0377
Amp0.001+Psd0.001+Ang0.01	22+3+7	98.11	96.82	0.96	0.96	0.0215	0.0212
Amp0.005+Psd0.0001+Ang0.05	10+9+2	96.59	94.46	0.92	0.91	0.0415	0.0430
Amp0.002+Psd0.00001+Ang0.005	16+31+11	96.60	95.72	0.94	0.94	0.0334	0.0327
Amp0.0005+Psd0.00001+Ang0.004	32+31+15	97.34	97.66	0.96	0.96	0.0242	0.0249
Amp0.0006+Psd0.000005+Ang0.003	28+49+20	96.32	97.34	0.96	0.96	0.0252	0.0255
Amp0.0007+Psd0.00001+Ang0.0035	25+31+16	96.64	96.36	0.94	0.94	0.0315	0.0316
Amp0.0004+Psd0.00001+Ang0.003	34+31+20	96.80	98.12	0.95	0.95	0.0255	0.0248
Amp0.001+Psd0.001+Ang0.05	22+3+2	98	98.33	0.96	0.96	0.0211	0.0224
Amp0.002+Psd0.00001	*16+31*	*96.56*	*99.64*	*0.97*	*0.96*	*0.0185*	*0.0217*
Amp0.001+Psd0.001	22+3	97.01	97.32	0.95	0.95	0.0248	0.0263
Amp0.005+Psd0.0001	10+9	96.24	92.18	0.91	0.91	0.0466	0.0473
Amp0.0004+Psd0.00001	34+31	98.09	97.55	0.97	0.97	0.0187	0.0191
Amp0.0006+Psd0.000005	28+49	97.99	97.34	0.97	0.96	0.0191	0.0204
Amp0.0005+Psd0.00001	32+31	98.42	97.52	0.96	0.97	0.0207	0.0213
Amp0.001+Psd0.001+Ang0.001	22+3+61	93.77	97.43	0.93	0.93	0.0374	0.0377

Figure 2.3: Results of selecting and testing by neural Networks [8].

In summary for these techniques, regarding the result that they got and the accuracy of the model that they have built is quite impressive the obvious disadvantages of this method can be divided into 4 aspects: scalability,deployment,inference and cost of realization. In terms of deployment of the model, it will be incredibility complicated since if anyone wants to use it need to build this system from scratch. The only re-usable part is the trained Neural Network but for features extraction. So, you can only use the same autistic system and the cost of realization can't be estimated exactly. Compared to this model, the system we are going to present later in this thesis is more adaptable.

Almonds classification using supervised learning methods

The title of this section is the title of a paper published by Elila Halac, Emir Sokic, Emir Turajlic from the Faculty of Electrical Engineering Sarajevo. So, in order to be able to distinguish among different types of almonds, training sets/images for supervised learning methods needed to be generated. Therefore, various species of almonds are chosen, captured by a digital camera, sorted and labeled so that they may be easily identified. The base consists of a total of 110 images sorted in five classes (raw, blanched, roasted, roasted blanched, unknown/hazelnuts) with three states (whole, parted, and damaged). All images are captured under equal conditions (same digital camera, same position, same background). In terms of image pre-processing and features extraction they have used 4 techniques to segment the image: Region-based methods, Edge-based methods, Fuzzy Clustering and K means clustering. After that they based their study on 5 main extracted shape features: Eccentricity, Circularity, Major Axis Length, Radius, Roundness and one color feature after that Then they applied a PCA (Principle component analysis) in order to reduce less efficient/unused features. After that the model was followed by a classifier Either a Simple Fully Collected layer neural network or a SVVM (Support Vector Machine). Figure 2.4 will summarise their results[1].

Classification method	Result (%)	Corr. img.	Avg. [ms]/img.
SVM Hybrid	92.7273	51	5.5
SVM One vs One	87.2727	48	22.8
Hybrid NN	85.4545	47	177.1
NN with N classes	72.7273	40	10.9
NN with M cl. P states	70.9091	39	10.9
SVM One vs All	43.6364	24	4.2

Figure 2.4: Showing the results of the used methods [1]

As summary/criticism for this model, despite the accuracy of the model that they have got, there are few weaknesses|flaws that need to be raised about this model. Primarily, it's a complex model, it has many phases in it (we mentioned around 5 in the explanation before) and most of them requires human intervention except the final phase which is the classification. Due to that, at the serving|inference time it would require the same level of intervention.

Chapter 3

Data collection and pre-Proccesing

In this chapter, we will be discussing the first step of the project and perhaps the hardest part which is the collection and labeling of the dataset and then pre-processing it to be able to feed it to the classifier .

3.1 Data Gathering and labeling

In order to be able to distinguish among different types of almonds, training sets/images for supervised learning methods needed to be generated. Therefore, various species of almonds are chosen, captured by a digital camera, sorted and labeled so that they may be easily identified. The base consists of a total of around 1000 images sorted in sixteen classes equally (1-tardy nonpareil , 2-fra gulio grade , 3-atocha , 4-picantil 5-pri morsky , 6-peerless , 7-moncaio , 8-marta , 9-ferralise , 10-garrigaes , 11-masbovera , 12-ai , 13-planeta , 14-texas , 15-philys , 16-desmayo lagueto). All images are captured under equal conditions (same Smart phone camera Honor 8x, same background) although the position

was taking randomly so any class had multiple random positions in order to give our model a better generalization. The dataset structure is represented by figure 3.1, while the stacked different classes from the dataset are shown in Figure 3.2. The entire dataset can be downloaded from the following URL: `https://drive.google.com/drive/folders/13xUelFiqSXQYVYQrcIDyVWmTKp6owBH7?usp=sharing`

Class Name	Number of images collected
1-tardy nonpareil	152
2-fra gulio grade	213
3-atocha	193
4-picantil	171
5-pri morsky	161
6-peerless	153
7-moncaio	155
8-marta	182
9-ferralise	304
10-garrigaes	157
11-masbovera	202
12-ai	195
13-planeta	180
14-texas	252
15-philys	161
16-desmayo lagueto	215

Figure 3.1: Classes with number of data collected per each

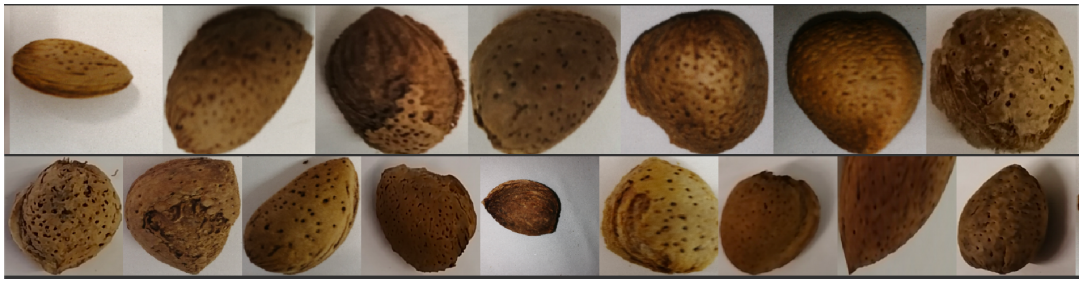


Figure 3.2: Stacked images of all the different classes

In the collection process many consideration were taken into account from which we can mention the popular generalisation problem that happens when you train learners (including humans) so once you train the model on a training set and got a result (if there is a pattern) it would be challenging to get the same result at the testing set when some of the domain features will change. As an example, in our case, is the face (position) of the almond unit since it's almost a ball it has an very large number of faces so by taking an image of an almond using just one position would cause a miss-match at the testing time. For that matter we have decided to introduce entropy in the data-collection by taking as many positions (angles) as possible in order to make the model generalise better at the testing set. Following up on the generalisation problem the back-ground also was chosen carefully when doing this phase, so all the collected images had a white background. Additionally lighting conditions too some images were taken using flash and some of them not .

The data set was collected from Institute Polytechnico Bragança at the agriculture School by the help and assistance of staff of the **Laboratório de Agrobiotecnologia (Professors Nuno Rodrigues and José Alberto Pereira)**

3.2 Image pre-processing and feature extraction

In real life a huge amount of data can be collected in order to tackle a specific problem, the catch is how to extract only the necessary (useful) features that help to solve the problem. Manual features extraction has shown that it's quite limited when dealing with complex/hidden features. So the solution is to develop relatively automatic processes and techniques in order to solve that. Feature extraction is a part of the dimensionality reduction process in which an initial set of the raw data is divided and reduced to more manageable groups. So when you want to process it will be easier. A main and most important characteristic of these large data sets is that they have a large number of variables. These variables require a lot of computing resources to process them. So, feature extraction basically helps to get the best feature from those big data sets by select and/or combine variables into features, effectively reducing the amount of data. These features are easy to process, but still able to describe the actual data set with accuracy and originality. The technique of extracting the features is useful when you have a huge dataset (whether in the number of rows or columns) and we want to avoid redundant data and reduce the number of resources without losing important or relevant information [9] .

In the past, the feature extractors were often hand-crafted. The problem with this approach is that we do not always know in advance, which features are interesting. The trend in machine learning has been towards learning the feature detectors as well, which enables using the complete data [10] .

Prior to performing feature extraction, image segmentation needs to be conducted. Different methods of image segmentation may be applied, such as:

Threshold methods, Region-based methods, Edge-based methods, Fuzzy Clustering, and one of the most popular methods being K- means clustering algorithm. K-means clustering algorithm is an unsupervised algorithm that may be used to segment the area of interest from the background. The algorithm consists of two separate phases. In the first phase, it computes k centroids randomly, where k is given in advance. In the second phase, it joins each point to the cluster which has the smallest distance from the centroid. The distance between points and centroid may be computed using different methods and the most popular is Euclidean distance. Since every image has one almond and relatively homogeneous background, there are two clusters to classify. We used the faster implementation of K-means (using pre-allocation and parallel operations) to optimize algorithm time. The L a b space color is used for segmentation, where a and b components are used to differentiate almond from the background. As a result of clustering, there are two images; one of them is the segmented almond and the other one is the background. Figure 3.4 illustrates the process of segmentation using the example of the first class of the almond (tardy nonpareil).

3.2.1 Methods Used

In our process we have tried many manual thresh-holding techniques in order to segment the image and localize the almonds in the picture by denoising it and remove the background. The process can be divided into 4 major steps.

Bluring the image

In this phase, we have used OpenCV 2.0 functions to blur images. When we blur an image, we make the color transition from one side of an edge in the

image to another smooth rather than sudden. The effect is to average out rapid changes in pixel intensity. The blur, or smoothing, of an image, removes "outlier" pixels that may be noise in the image. Blurring is an example of applying a low-pass filter to an image. In computer vision, the term "low-pass filter" applies to removing noise from an image while leaving the majority of the image intact. A blur is a very common operation we need to perform before other tasks such as edge detection. There are several different blurring functions in the OpenCV 2.0 module, so we will focus on just one here, the Gaussian blur [11] . In a blur, we consider a rectangular group of pixels surrounding the pixel to filter. This group of pixels, called the kernel, moves along with the pixel that is being filtered. So that, the filtered pixel is always in the center of the kernel, the width and height of the kernel must be odd. In the example shown above, the kernel is square, with a dimension of seven pixels. To apply this filter to the current pixel, a weighted average of the color values of the pixels in the kernel is calculated. In a Gaussian blur, the pixels nearest the center of the kernel are given more weight than those far away from the center. This averaging is done on a channel-by-channel basis, and the average channel values become the new value for the filtered pixel. Larger kernels have more values factored into the average, and this implies that a larger kernel will blur the image more than a smaller kernel.

Using OpenCV 2.0 with python this operation can be done in two lines of code

```
# read image
show = cv2.imread(image, cv2.IMREAD_COLOR)
# applying the blur
imgBlur = cv2.GaussianBlur(show, (7, 7), 1)
```

Graying the Image

In this second phase, we have used OpenCV 2.0 functions to apply thresholding to an image. Thresholding is a type of image segmentation, where we change the pixels of an image to make the image easier to analyze. In thresholding, we convert an image from color or grayscale into a binary image, i.e., one that is simply black and white. Most frequently, we use thresholding as a way to select areas of interest of an image, while ignoring the parts we are not concerned with, we will use the masks returned by these functions to select the parts of an image we are interested in.

The process works like this: first, we will load the original image, blur it (as explained in the previous section) and then convert it to grayscale. Then, we will use the bigger than (>) operator to apply the threshold t, a number in the closed range [0.0, 1.0]. Pixels with color values on one side of t will be turned "on," while pixels with color values on the other side will be turned "off." In order to use this function, we have to determine a good value for t. How might we do that? Well, one way is to look at a grayscale histogram of the image using an OpenCV function when we run this code shown below you will get the histogram shown in figure 3.3.

```
# read image
show = cv2.imread(image, cv2.IMREAD_COLOR)
# Calculating the Histogram
hist = cv2.calcHist(images, channels, mask, histSize,
ranges[, hist[, accumulate]])
```

Some explanations about this code:

1. Channels: it is the index of a channel for which we calculate histogram. For a grayscale image, its value is [0] and for a color image, you can pass [0], [1], or [2] to calculate a histogram of blue, green, or red channel respectively.

2. Mask: mask image. To find a histogram of the full image, it is given as "None".

3. HistSize: this represents our BIN count. For full scale, we pass [256].

4. ranges : this is our RANGE. Normally, it is [0,256].

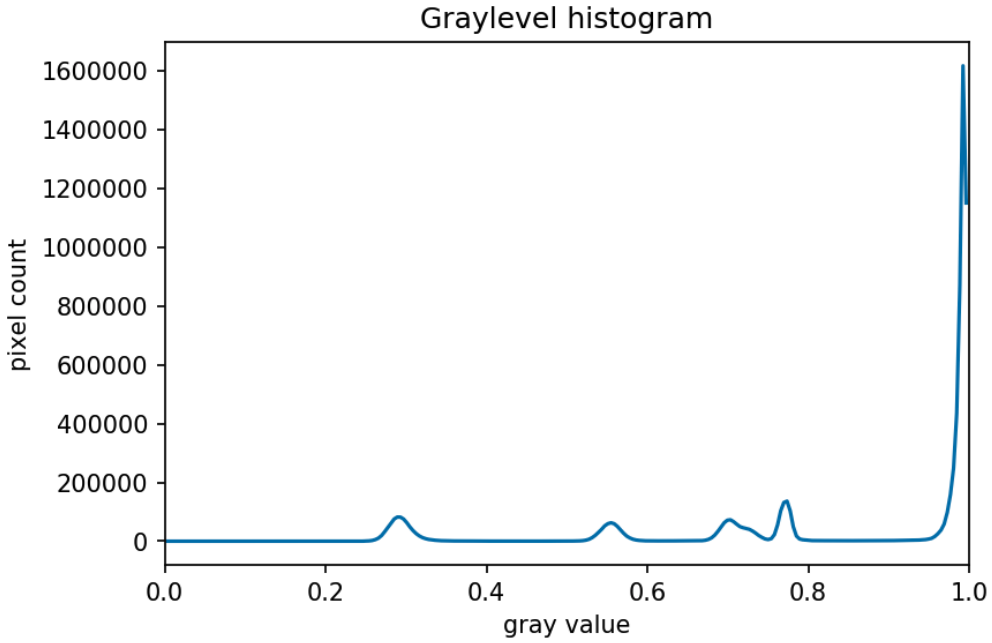


Figure 3.3: GrayScale histogram from an almonde image sampled from the dataset

Since the image has a white background, most of the pixels in the image are white. This corresponds nicely to what we see in the histogram: there is a spike near the value of 1.0. If we want to select the shapes and not the background,

we want to turn off the white background pixels, while leaving the pixels for the shapes turned on. So, we should choose a value of t somewhere before the large peak and turn pixels above that value "off".

One other way to do it is to create a simple Track Bar that has properties of the image such as (saturation, hue value ..) and customize it manually to extract the values that will help you to separate the object from the background.

Edge and Shape detection using Canny Dilation Algorithms to image

After getting our thresh-hold values whether using histograms or using the manual track-bar method we can now move to the next phase which is applying the filters using Canny dilate. So, we first separated the image from the background and then we gonna use this function to detect the edges/shape of the object. Canny edge detection is a technique to extract useful structural information from different vision objects and dramatically reduce the amount of data to be processed. It has been widely applied in various computer vision systems. Canny has found that the requirements for the application of edge detection on diverse vision systems are relatively similar. Thus, an edge detection solution to address these requirements can be implemented in a wide range of situations. The general criteria for edge detection include:

1. Detection of edge with low error rate, which means that the detection should accurately catch as many edges shown in the image as possible;

2. The edge point detected from the operator should accurately localize on the center of the edge;

3. A given edge in the image should only be marked once, and where possible, image noise should not create false edges.

To satisfy these requirements Canny used the calculus of variations – a technique that finds the function which optimizes a given functional. The optimal function in Canny's detector is described by the sum of four exponential terms, but it can be approximated by the first derivative of a Gaussian.

Among the edge detection methods developed so far, the Canny edge detection algorithm is one of the most strictly defined methods that provide good and reliable detection. Owing to its optimally to meet with the three criteria for edge detection and the simplicity of process for implementation, it became one of the most popular algorithms for edge detection [12] .

For the Second Technique which is Dilation, it is one of the basic operations in mathematical morphology. Originally developed for binary images, it has been expanded first to gray-scale images, and then to complete lattices. The dilation operation usually uses a structuring element for probing and expanding the shapes contained in the input image [13].

These operation can be applied using the following code

```
#Graying the image
imgGray = cv2.cvtColor(imgBlur, cv2.COLOR_BGR2GRAY)

# edge detection
imgCanny = cv2.Canny(imgGray, threshold1, threshold2)

# Shape detection
kernel = np.ones((5, 5))
imgDil = cv2.dilate(imgCanny, kernel, iterations=1)
```

What this code does is basically first applies the gray filter and then uses

the Canny function with the gray image as an input and two other parameters which are the two thresh-holds values extracted using the previous track-Bar Method. After getting the cannied Image, we pass it to the dilate function giving the kernel of 5*5 as argument which is used for structuring element used for dilation. If element=Mat(), a 3 x 3 rectangular structuring element is used. The kernel can be created using getStructuringElement Function.

Find the contours of the image

The last step and the final stage of the processing is to get the contours of the object (almond in the image). Contouring is the process of identifying structural outlines of objects in an image which in turn can help us identify the shape of the object [14]. Well, when we perform edge detection, we find the points where the intensity of colors changes significantly, and then we simply turn those pixels on. However, contours are abstract collections of points and segments corresponding to the shapes of the objects in the image. As a result, we can manipulate contours in our programs such as counting the number of contours, using them to categorize the shapes of objects, cropping objects from an image (image segmentation), and much more. Contour detection is not the only algorithm for image segmentation though, there are a lot of others, such as the current state-of-the-art semantic segmentation, hough transform, and K-Means segmentation.

To perform this operation we execute this line of code:

```
# read image
show = cv2.imread(image, cv2.IMREAD_COLOR)
# get all the contours in the image
contours, hierarchy = cv2.findContours(imgDil, cv2.RETR_EXTERNAL,
```

```
cv2.CHAIN_APPROX_NONE)
```

The findContours function of OpenCV 2.0 uses by default the SATODHI SUZUK Algorithm. This was one of the first algorithms that define the hierarchical relationships among the borders. This algorithm also differentiates between the outer boundary or the hole boundary. The steps of the algorithm are discussed below:

1. Step-1 If it's an outer border (i.e. fij = 1 and fi,j-1 = 0) then increment the NBD and set (i2, j2) as (i, j-1). Else if it is a hole border, increment NBD. Set (i2, j2) as (i, j+1) and LNBD = fij in case fij >1. Otherwise, go to step 3 [15].

2. Step-2 Now, from this starting point, we will trace the border. This can be done as:

   Starting from (i2, j2) look around clockwise the pixels in the neighborhood of (i, j) and find a nonzero pixel and denote it as (i1, j1). If no nonzero pixels are found, set fij = -NBD and go to step 4. Set (i2, j2) = (i1, j1) and (i3, j3) = (i,j). Starting from the next element of the pixel (i2, j2) in the counterclockwise order, again traverse the neighborhood of the (i3, j3) in the counterclockwise direction to find the first nonzero pixel and set it to (i4, j4). Change the value of the current pixel (i3, j3) as if the pixel at (i3, j3 +1) is a 0-pixel belonging to the region outside the boundary, set the current pixel value to -NBD. if the pixel at (i3, j3 +1) is not a 0-pixel and the current pixel value is 1, set the current pixel value to NBD. Otherwise, do not change the current pixel value. if in step 2.3, we return to the starting point again i.e (i4, j4) = (i, j) and (i3, j3) = (i1, j1) go to step 3. Otherwise, set (i2, j2) = (i3, j3) and (i3, j3) = (i4, j4) and go back and restart the step 2 [15].

3. Step-3 If fij != 1 then set LNBD = |fij| and start scanning from the next pixel (i, j+1). Stopping criteria is when we reached the bottom right corner of the image [15].

After we applied the function to find contours we just take the biggest one in the image (Of course there are other methods to extract the more relevant contour but for our case, this one worked fine) and then we corp the image using those coordinates. The following code will do so:

```
# get all the contours in the image
contours, hierarchy = cv2.findContours(imgDil, cv2.RETR_EXTERNAL
s, cv2.CHAIN_APPROX_NONE)

# use only the biggest contour/object
for cnt in contours:
    area_contours.append(cv2.contourArea(cnt))
contour_used = contours[np.argmax(area_contours)]

# corp the image using the Biggest contour areea
x, y, w, h = cv2.boundingRect(contour_used)
resized_image = cv2.resize(show[y:y + h, x:x + w], (nrows, ncolumns)
, interpolation=cv2.INTER_CUBIC)
stacked_images = stack_images(0.8, [show
, imgBlur, imgGray, imgCanny, imgDil, resized_image] + drawn_contours)
cv2.imshow("stack images ", stacked_images)
cv2.waitKey(0)
return resized_image
```

3.3 Conclusion of the final results of the segmentation

The images shown in 3.4 are the results of each step of the processing/segmentation applied in our project starting with the original picture at the left going to the blurred one beside it and then the grayed one from there you apply dialect and canny consecutively. Finally you extract the contours and corp the image.

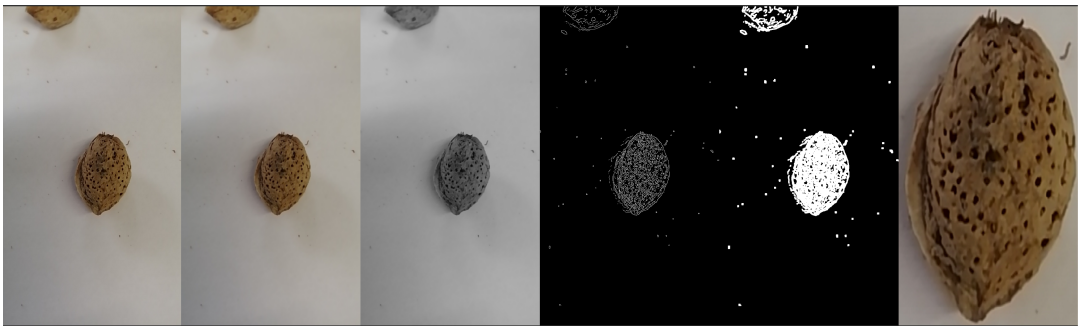


Figure 3.4: Stacked image after segmentation

Chapter 4

Deep Learning Based Classification

In this chapter, we will be discussing the Deep learning model that we have built and all the inspiration behind Convolutional Neural Networks as well the Material(software/hardware) used in development phase.

4.1 Material Used

4.1.1 Hardware

The implementation environment was a MacbookPro laptop computer with an Intel Core i9-7210 3.70 GHz CPU, 16 GBs of RAM, and NVIDIA GeForce 840M GPU.The Final model training was done on the google cloud platform in **AI Platform Training and Prediction API** instance type. The Dataset collection was done by a smartphone Honor8x Camera 20 MP, f/1.8, 27 mm (wide), PDAF, 2 MP, (depth) with LED flash, panorama, HDR.

4.1.2 Software

Image Processing tools

The Framework used for all the image processing/segmentation detailed in the previous part was done using Python OpenCV 2.0, OpenCV (Open Source Computer Vision Library), originally developed by Intel in 2000, is a multi-platform library, totally free for academic and commercial use, for the development of applications in the area of Computer Vision, simply following the BSD license model Intel. OpenCV has Image and Video I/O Processing, Data Structure, Linear Algebra, Basic GUI (Graphical User Interface) modules with independent window system, mouse and keyboard control, in addition to more than 350 computer vision algorithms such as image filters, camera calibration, object recognition, structural analysis, and others. Its image processing is in real time. This library was developed in the C / C ++ programming languages. It also supports programmers who use Java, Python, and Visual Basic and want to incorporate the library into their applications. Version 1.0 was released in late 2006 and 2.0 was released in September 2009 [16].
As for our solution require some Numeric Calculation Numpy, the Python liberty was used for that matter.NumPy offers comprehensive mathematical functions, random number generators, linear algebra routines, Fourier transforms, and more. And also in terms of performance, the core of NumPy is a well-optimized C code. Enjoy the flexibility of Python with the speed of compiled code.

Deep learning Framework

TensorFlow is unarguably one of the most popular deep learning frameworks. Developed by the Google Brain team, TensorFlow supports languages such as

Python, C++, and R to create deep learning models along with wrapper libraries. It is available on both desktop and mobile.

The most well-known use case of TensorFlow has got to be Google Translate coupled with capabilities such as natural language processing, text classification, summarizing, speech/image/handwriting recognition, forecasting, and tagging [17].

TensorFlow's visualization toolkit, TensorBoard, provides effective data visualization of network modeling and performance.

TensorFlow Serving, another tool of TensorFlow, is used for the rapid deployment of new algorithms/experiments while retaining the same server architecture and APIs. It also provides integration with other TensorFlow models, which is different from the conventional practices and can be extended to serve other models and data types.

TensorFlow is one of the most preferred deep learning frameworks as it is Python-based, supported by Google, and comes loaded with top-notch documentation and walkthroughs to guide you.

Highlights of TensorFlow :

1. Robust multiple GPU support;

2. Graph visualization and queues using TensorBoard;

3. Known to be complex and has a steep learning curve;

4. Excellent documentation and community support.

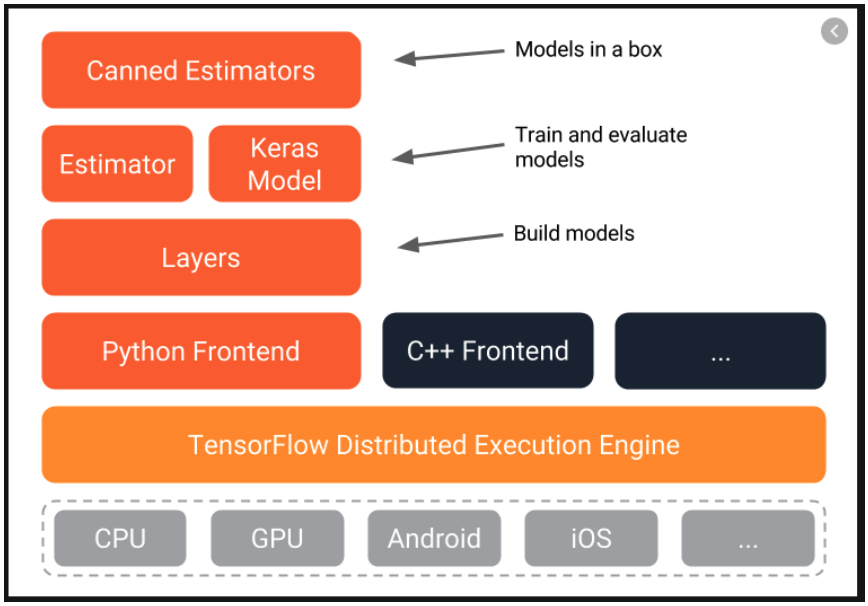


Figure 4.1: Tensorflow EcoSystem [17].

As a comparison to the framework that we have chosen, Pytorch is perhaps also one of the popular frameworks out there. It was developed by Facebook and was first publicly released in 2016. It was created to offer production optimizations similar to TensorFlow while making models easier to write figure 4.1 shows tensorflow ecoSystem.

1. **Debugging** Since computation graph in PyTorch is defined at run-time you can use our favorite Python debugging tools such as pdb, ipdb, PyCharm debugger or old trusty print statements. This is not the case with TensorFlow. You have an option to use a special tool called tfdbg which allows to evaluate tensorflow expressions at runtime and browse all tensors and operations in session scope. Of course, you won't be able to debug any python code with it, so it will be necessary to use pdb separately [18].

2. **Visualization** Tensorboard is awesome when it comes to visualization . This tool comes with TensorFlow and it is very useful for debugging and

comparison of different training runs. For example, consider you trained a model, then tuned some hyperparameters and trained it again. Both runs can be displayed at Tensorboard simultaneously to indicate possible differences. Tensorboard can: display model graph , plot scalar variables , visualize distributions and histograms, visualize images, visualize embedding or play audio. Tensorboard competitor from the PyTorch side is visdom. It is not as feature-complete, but a bit more convenient to use. Also, integrations with Tensorboard do exist. Nevertheless you are free to use standard plotting tools such as matplotlib and seaborn [18].

3. **Deployment** If we start talking about deployment TensorFlow is a clear winner for now: is has TensorFlow Serving which is a framework to deploy your models on a specialized gRPC server. Mobile is also supported. When we switch back to PyTorch we may use Flask or another alternative to code up a REST API on top of the model. This could be done with TensorFlow models as well if gRPC is not a good match for your use case. However, TensorFlow Serving may be a better option if performance is a concern. Tensorflow also supports distributed training which PyTorch lacks for now [18].

4.2 Deep learning Model

4.2.1 Convolution models

Convolutional networks, also known as convolutional neural networks, or CNNs, are a specialized kind of neural network for processing data that has a known grid-like topology. Examples include time-series data, which can be thought of as a 1-D grid taking samples at regular time intervals, and image data, which

can be thought of as a 2-D grid of pixels. Convolutional networks have been tremendously successful in practical applications. The name "convolutional neural network" indicates that the network employs a mathematical operation called convolution. Convolution is a specialized kind of linear operation. Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers [19].

Convolutional networks are perhaps the greatest success story of biologically inspired artificial intelligence. Though convolutional networks have been guided by many other fields, some of the key design principles of neural networks were drawn from neuroscience. The history of convolutional networks begins with neuroscientific experiments long before the relevant computational models were developed. Neurophysiologists David Hubel and Torsten Wiesel collaborated for several years to determine many of the most basic facts about how the mammalian vision system works [3]. Their accomplishments were eventually recognized with a Nobel prize. Their findings that have had the greatest influence on contemporary deep learning models were based on recording the activity of individual neurons in cats. They observed how neurons in the cat's brain responded to images project precise locations on a screen in front of the cat. Their great discovery was that neurons in the early visual system responded most strongly to very specific patterns of light, such as precisely oriented bars, but responded hardly at all their patterns [3].

4.2.2 Model Implementation (Design And training)

RGB based with Fully Connected Dense Network

In the first iteration of defining our model, we have started by using a dense network to be trained only on RGB (Red Green Blue) of the image, so at the

beginning, we read the image per class and I have used the function shown below to calculate the mean value of each 3 channel of the image:

```
# Function Definition
def get_mean_RGB(img):
    return np.mean(img[:, :, 0]), np.mean(img[:, :, 1]),
    np.mean(img[:, :, 2])
# read image
show = cv2.imread(image, cv2.IMREAD_COLOR)

# Extract the MEAN RGB of each channel
mean_r,mean_g,mean_b = get_mean_RGB(show)
```

And by the same way, after calculating the mean RGB of each image, we built a dense network consisting of 3 input Neurons, 16 as output neurons (the classes are encoded using the one-hot encoding method), and 4, 5 Neurons in the hidden layers the figure 4.2 and figure 4.3 shows the model definition parameters as well as the visualization of the model using tensorboard.

```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 3)                 0
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 16
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 25
=================================================================
Total params: 41
Trainable params: 41
Non-trainable params: 0
_________________________________________________________________
acc: 40.00%
```

Figure 4.2: Model definition (parameters , architecture)

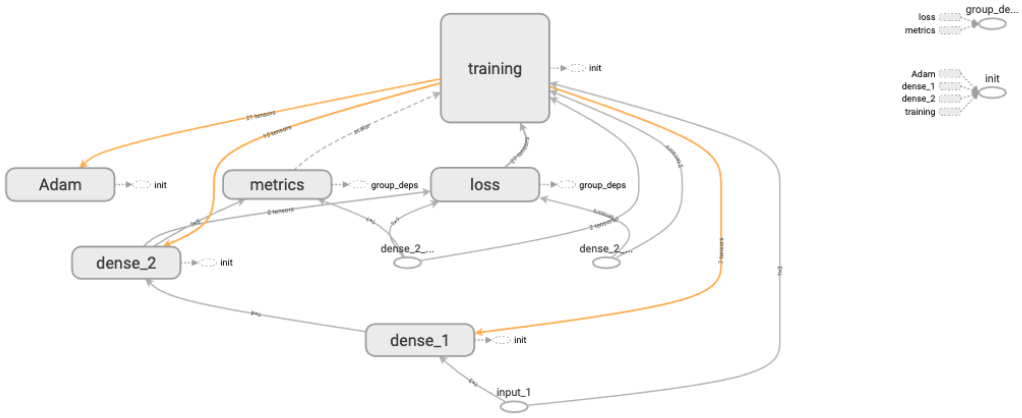


Figure 4.3: The visualisation of the model using tensorboard

After training this model on just 5 classes out of the 16 classes that we have and using all the images, round 100 per each class and training around 10000 iterations, we got poor results. Despite many tuning by changing the number of layers /neurons and also by changing the optimizers, result iterations seemed to stay the same. We evaluated the model using the Accuracy Metric (Default in TensorFlow) that creates two local variables, total and count that are used to compute the frequency with which y-pred matches y-true. This frequency is ultimately returned as categorical accuracy: an idempotent operation that simply divides the total by count.

Apparently the color is not enough to distinguish between the almonds so we have to dig deeper and extract more relevant features. The figure 4.4. Below the accuracy curve during the training is shown:

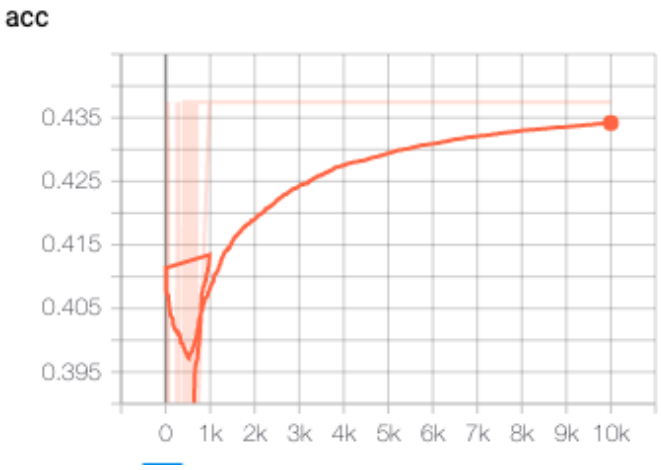
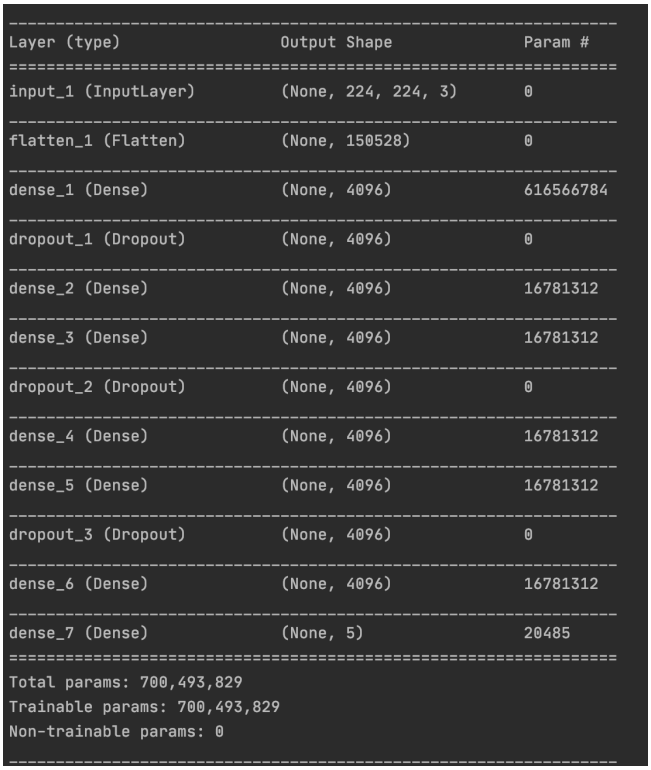


Figure 4.4: Accuracy curve during the training

Image based with Fully Connected Dense Network

In the second iteration of the implementation after we realized that using just the color of the object is not enough to classify it, we have implemented a Dense network consisting of 50176 input layer after applying the flattening of the image using the flatten layer of Keras. It consists on 6 Dense layers as hidden layers around 4096 neurons each and by 5 output softmax neurons according to the 5 classes that we have. Figure 4.5 below shows the structure of the network along with the number of trainable parameters per each layer.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 150528)            0
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              616566784
_________________________________________________________________
dropout_1 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_2 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dense_3 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dropout_2 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_4 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dense_5 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dropout_3 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_6 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dense_7 (Dense)              (None, 5)                 20485
=================================================================
Total params: 700,493,829
Trainable params: 700,493,829
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.5: Structure of the model with the number of parameters

As said, the model was trained using just 5 classes out of the 16 and all the steps of the pre-processing described in the previous part have been applied. The figure 4.6 show a visualization of the model using tensorboard.

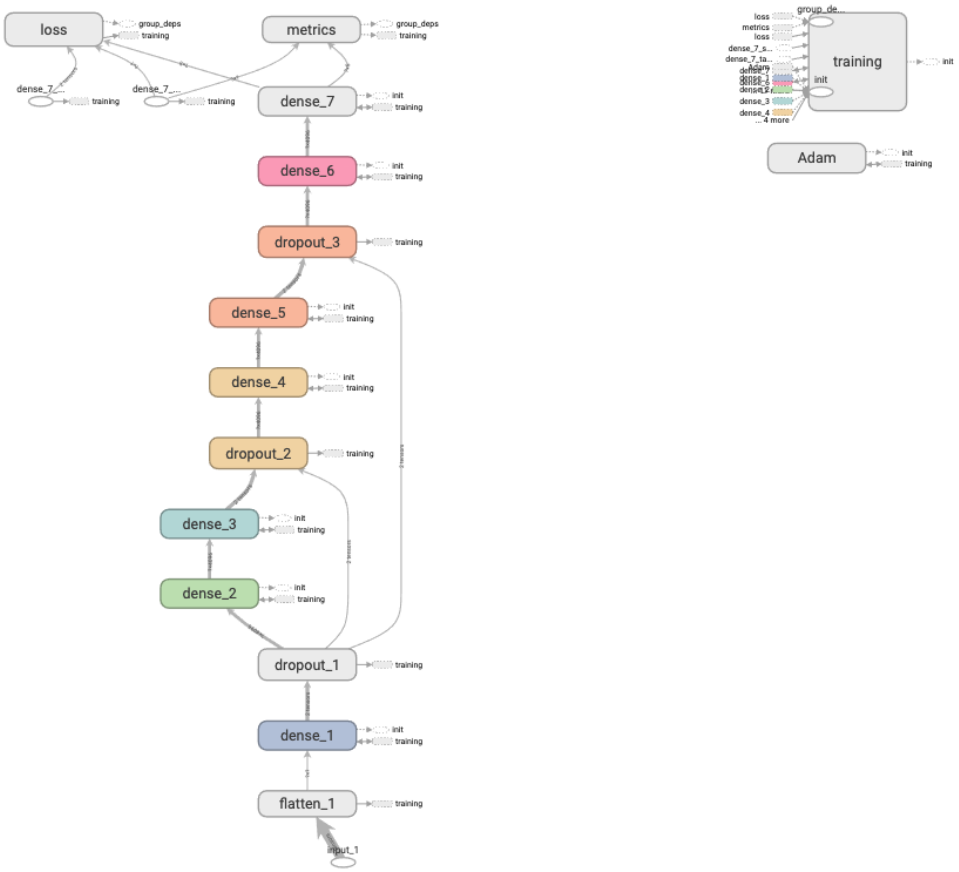


Figure 4.6: The visualisation of the model using tensorboard

In this second iteration using the full features of the image after training this model on just 5 classes out of the 16 classes that we have and using all the images round 100 per each class and it was trained around 10000 iterations we got poor results. The accuracy seems to be stuck at 70 percent which is better compared to the first iteration but not enough as a final solution. Once again, we evaluated the model using the Accuracy Metric (Default in TensorFlow) that creates the two local variables, total and count, used to compute the frequency with which y-pred matches y-true. This frequency is ultimately returned as

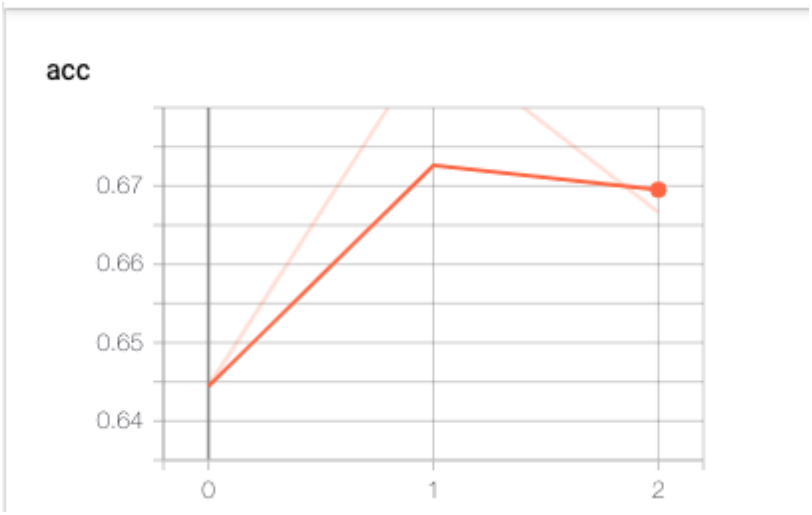


Figure 4.7: Accuracy curve During the training

categorical accuracy: an idempotent operation that simply divides the total by count (Figure 4.7).

The image-based training seems promising compared to the first iteration, that's why we gonna continue down this road by just changing the model from a simple ANN to a convolutional NeuralNetwork as we will show in the next section.

Image Bases with AlexNet V1

In the third iteration of the development and after we got a relatively good result using the features of the image compared to using just the color, we decided to use a convolutional based model to process/classify the almonds.
The Model Architecture was based on ImageNet model September 10, 2012. For a bit of context, ImageNet is a large visual database designed for use in visual object recognition software research. More than 14 million images have been hand-annotated by the project to indicate what objects are pictured and

in at least one million of the images, bounding boxes are also provided. ImageNet contains more than 20,000 categories with a typical category, such as "balloon" or "strawberry", consisting of several hundred images. The database of annotations of third-party image URLs is freely available directly from ImageNet, though the actual images are not owned by ImageNet since 2010, the ImageNet project runs an annual software contest, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), where software programs compete to correctly classify and detect objects and scenes. The challenge uses a "trimmed" list of one thousand non-overlapping classes [19].

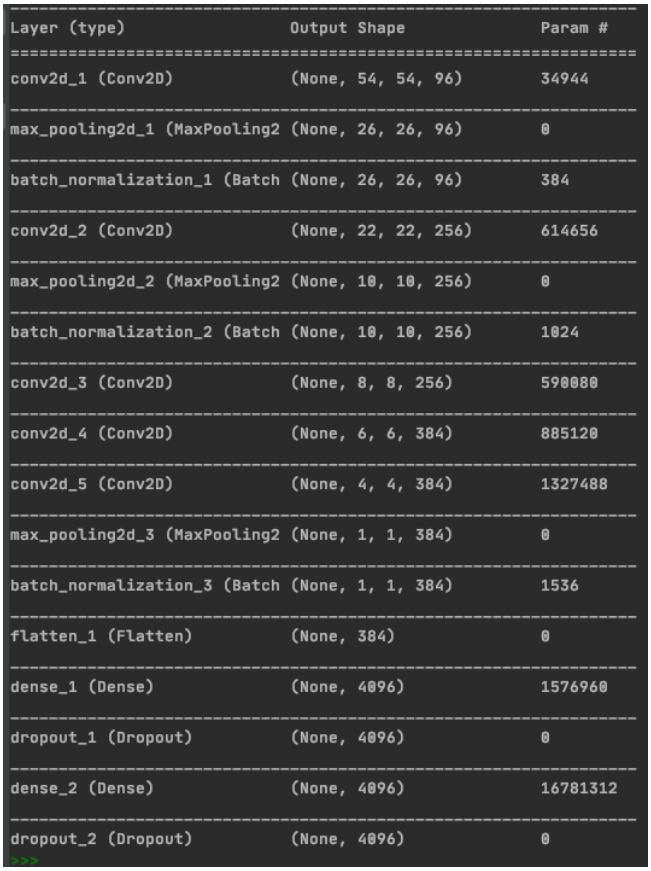
The architecture of our network is summarized in Figure 4.8. It contains eight learned layers —five convolutional and three fully-connected. The Remarkable things about the model architecture compared to its predecessors is the activation function where before this model the standard Activation Function was the Tanh function given by:

$$f(x) = (1 + e^{-x})^{-1} \tag{4.1}$$

In this model, it was changed to a non-saturating non-linearity called ReLU function given by:

$$f(x) = max(0, x) \tag{4.2}$$

Deep convolutional neural networks with ReLUs train several times faster than their equivalents with Tanh units.



Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 54, 54, 96)	34944
max_pooling2d_1 (MaxPooling2	(None, 26, 26, 96)	0
batch_normalization_1 (Batch	(None, 26, 26, 96)	384
conv2d_2 (Conv2D)	(None, 22, 22, 256)	614656
max_pooling2d_2 (MaxPooling2	(None, 10, 10, 256)	0
batch_normalization_2 (Batch	(None, 10, 10, 256)	1024
conv2d_3 (Conv2D)	(None, 8, 8, 256)	590080
conv2d_4 (Conv2D)	(None, 6, 6, 384)	885120
conv2d_5 (Conv2D)	(None, 4, 4, 384)	1327488
max_pooling2d_3 (MaxPooling2	(None, 1, 1, 384)	0
batch_normalization_3 (Batch	(None, 1, 1, 384)	1536
flatten_1 (Flatten)	(None, 384)	0
dense_1 (Dense)	(None, 4096)	1576960
dropout_1 (Dropout)	(None, 4096)	0
dense_2 (Dense)	(None, 4096)	16781312
dropout_2 (Dropout)	(None, 4096)	0

>>>

Figure 4.8: Structure of the model with the number of parameters

Once again, the model was trained using just 5 classes out of the 16 and all the steps of the pre-processing in the previous part have been applied. Figure 4.9 below shows a visualization of the model using tensorboard.

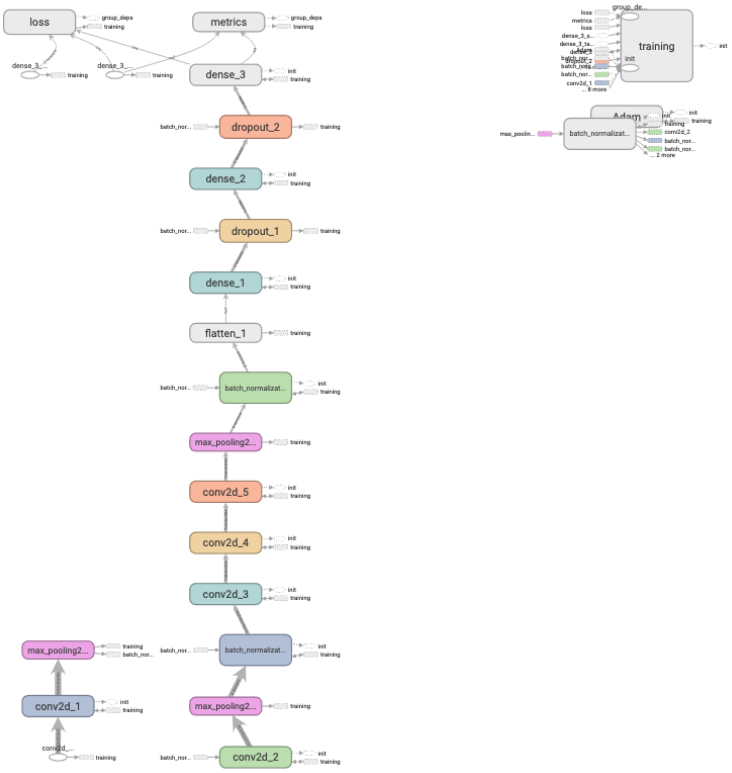


Figure 4.9: Visualisation of the model using tensorboard

In this third iteration of the development, we trained our models using stochastic gradient descent with a batch size of 8 examples with a 10 percent validation set and on 10000 iterations. We initialized the weights in each layer from a zero-mean Gaussian distribution with a standard deviation of 0.01. We initialized the neuron biases in the second, fourth, and fifth convolutional layers, as well as in the fully-connected hidden layers, with the constant 1. This initialization accelerates the early stages of learning by providing the ReLUs with positive inputs. We initialized the neuron biases in the remaining layers with the constant 0.

Our results on the almonds dataset using this model was as follows Accuracy: 89 percent. The figure 4.10 below shows the accuracy and the loss curve during

the training. The training lasted more than 5 days.

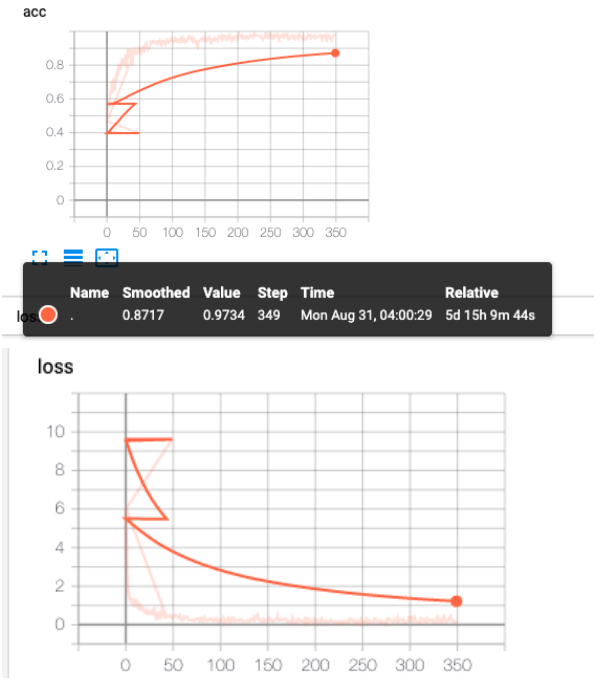


Figure 4.10: Loss and accuracy of V1 AlexNet

Image Bases with AlexNet V2

Given the impressive results of the previous model compared to the Dense network beforehand, we have decided to follow up on that by optimizing the AlexNet both in terms of accuracy and training time (which the V1 of it took more than five days to get trained). So, in this final iteration of the implementation, we altered the architecture of Alexnet by making it smaller by removing neurons and also layers. After considerable trials we arrived at this architecture which was optimal at the time. We have basically removed some of the hidden layers after we realized that the model is taking too long in training due to it's bigger size. So by removing some of the hidden layers (both convolutional and dense) the result got improved and with way less training time. The model details are explained in figure 4.11 and 4.12.

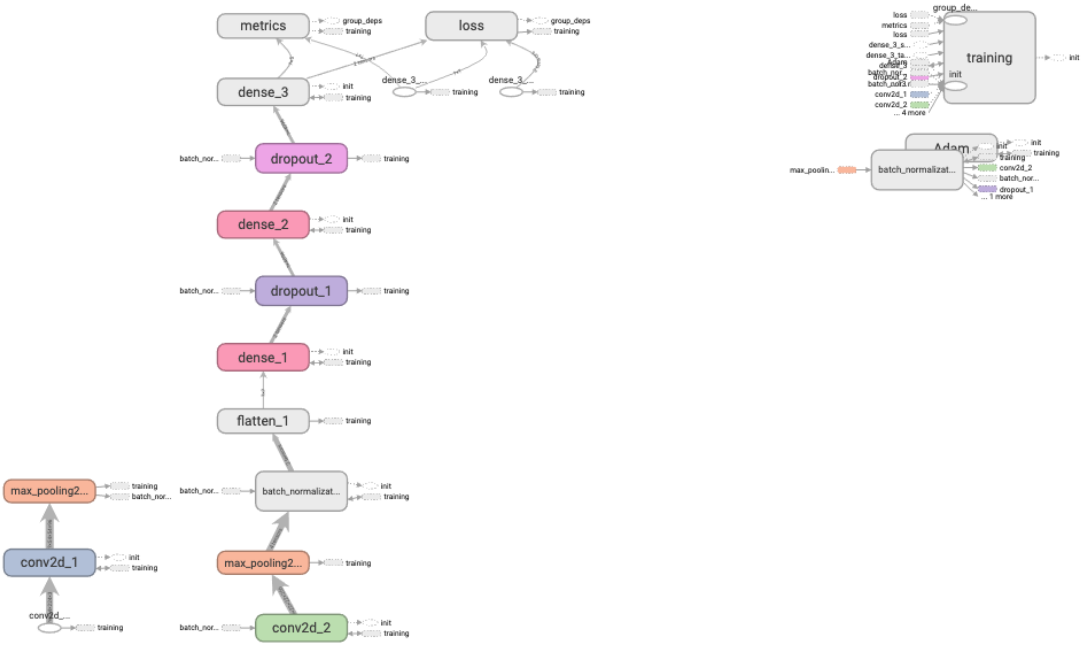


Figure 4.11: Visualisation of the model

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 54, 54, 96)	34944
max_pooling2d_1 (MaxPooling2	(None, 26, 26, 96)	0
batch_normalization_1 (Batch	(None, 26, 26, 96)	384
conv2d_2 (Conv2D)	(None, 22, 22, 256)	614656
max_pooling2d_2 (MaxPooling2	(None, 10, 10, 256)	0
batch_normalization_2 (Batch	(None, 10, 10, 256)	1024
flatten_1 (Flatten)	(None, 25600)	0
dense_1 (Dense)	(None, 4096)	104861696
dropout_1 (Dropout)	(None, 4096)	0
dense_2 (Dense)	(None, 4096)	16781312
dropout_2 (Dropout)	(None, 4096)	0
dense_3 (Dense)	(None, 5)	20485

Total params: 122,314,501
Trainable params: 122,313,797
Non-trainable params: 704

Figure 4.12: Model Architecture with details of the parameters

The result of this model was 94 percent on the testing set with the same iterations and batch size as the previous one 10000 and 8, using the same optimizers ADAM and it was trained on 16 classes (Figure 4.13).

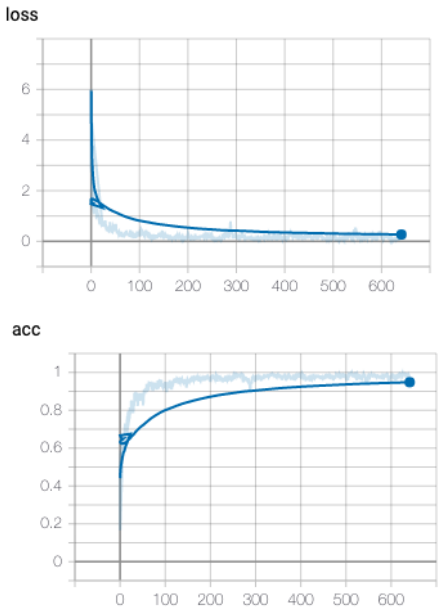


Figure 4.13: Results of the v2 Alexnet

4.3 Result and discussion

In this section, we will analyze our results further and discuss some of the model tuning techniques that we used, with references to the literature, how convolutional neural networks can be refined.

Successfully applying deep learning techniques requires more than just a good knowledge of what algorithms exist and the principles that explain how they work. A good machine learning practitioner also needs to know how to choose an algorithm for a particular application and how to monitor and respond to feedback obtained from experiments to improve a machine learning system. During day to day development of machine learning systems, practitioners need

to decide whether to gather more data, increase or decrease model capacity, add or remove regularizing features, improve the optimization of a model, improve approximate inference in a model, or debug the software implementation of the model. All of these operations are at the very least time-consuming to try out, so it is important to be able to determine the right course of action rather than blindly guessing. In the next subsection, we will be discussing this journey and why we chose what we have chosen [3].

4.3.1 Performance Metrics and model evaluation

Metric

Determining the goals of the experiment, in terms of which error metric to use, is a necessary first step because the error metric will guide all of the future actions. Note that in most applications, it is impossible to achieve absolute zero error. The Bayes error defines the minimum error rate that can hope to achieve, even if we have infinite training data and can recover the true probability distribution. This is because the input features may not contain complete information about the output variable, or because the system might be intrinsically stochastic and, of course, we will also be limited by having a finite amount of training data.

To evaluate the abilities of a machine learning algorithm, we had designed a quantitative measure of the performance of our model. Usually, performance measure P is specific to the task T being carried out by the system. In our task is **classification task** and the performance measure that we used is the **Accuracy**. Accuracy is just the proportion of examples for which the model produces the correct output. We can also obtain equivalent information by measuring the error rate, the proportion of examples for which the model produces

incorrect output. We often refer to the error rate as the expected 0-1 loss. The 0-1 loss on a particular example is 0 if it is correctly classified and 1 if it is not. The code below shows the function implemented in TensorFlow used for that matter :

```
# Function Definition
def test(X, Y, model_version):
try:
    model = keras.models.load_model('model/alexnet/saved_model/' +
    model_version)
    except (ImportError, IOError) as error:
    print("Error Loading model ", error)
else:
    print("Model Summary :")
    model.summary()
    scores = model.evaluate(X, Y, verbose=0)
    print((model.metricsNames[1], scores[1] * 100))
    return scores
```

where **model.metricsNames[1]** was defined in the model definition before start the training when we compiled the model, as this

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

Usually, we are interested in how well the machine learning algorithm performs on data that it has not seen before since this determines how well it will work when deployed in the real world. We, therefore, evaluate these performance measures using a test set of data that is separate from the data used for training the machine learning system.

However, many applications require more advanced metrics. Sometimes it is much more costly to make one kind of mistake than another. For example, an e-mail spam detection system can make two kinds of mistakes: incorrectly classifying a legitimate message as spam, and incorrectly allowing a spam message to appear in the inbox. It is much worse to block a legitimate message than to allow a questionable message to pass through. Rather than measuring the error rate of a spam classifier, we may wish to measure some form of the total cost, where the cost of blocking legitimate messages is higher than the cost of allowing spam messages.

Cross validation

Dividing the dataset into a fixed training set and a fixed test set can be problematic if it results in the test set being small. A small test set implies statistical uncertainty around the estimated average test error, making it difficult to claim that algorithm A works better than algorithm B on the given task. When the dataset has hundreds of thousands of examples like ours, this is not a serious issue. When the dataset is too small, alternative procedures enable one to use all the examples in the estimation of the mean test error, at the price of increased computational cost. These procedures are based on the idea of repeating the training and testing computation on different randomly chosen subsets or splits of the original dataset. To that matter, we used one of the most common of these is the k-fold cross-validation procedure, in which a partition of the dataset is formed by splitting it into k non-overlapping subsets. The test error may then be estimated by taking the average test error across k trials. On trial I, the i-th subset of the data is used as the test set, and the rest of the data is used as the training set. This can be implemented easily with Keras where you just have to pass validation split to the model.fit() function.

4.3.2 Default Baseline Models

After choosing performance metrics and goals, the next step in any practical application is to establish a reasonable end-to-end system as soon as possible. Depending on the complexity of the problem, it may even be possible to begin without using deep learning. If the problem has a chance of being solved by just choosing a few linear weights correctly, as a simple statistical model like logistic regression.

After observing the problem that we want to solve it clearly falls into an "AI-complete" category so it is obvious that simple models stated above are not gonna even approach the solution by any far that is why we have decided to go with a deep learning-based model. So as the first step we had to choose the general category of the model based on the structure of the data. Since our data is labeled it is clear that we are going to the user a model from the supervised learning bucket. After the next step is to examine whether our data (inputs) has a topological structure and the answer is yes, almost all images require extraction of it's topological structure in order to build a good classifier. Next is checked if our input or output has a sequence structure (latent representation of variable). In this case, the input is an image (matrices of pixels) and the output is a fixed label so there's no need to use sequence models such as gated recurrent net (LSTM or GRU) or it's variants. A reasonable choice of the optimization algorithm is SGD with momentum with a decaying learning rate (popular decay schemes that perform better or worse on different problems include decaying linearly until reaching a fixed minimum learning rate, decaying exponentially, or decreasing the learning rate by a factor of 2-10 each time validation error plateaus). Another very reasonable alternative which is the one we used at the end is Adam. Batch normalization can have a dramatic effect on optimization

performance, especially for convolutional networks and networks with sigmoidal nonlinearities. While it is reasonable to omit batch normalization from the very first baseline, it should be introduced quickly if optimization appears to be problematic.

These were the steps that we have followed to construct our solution. Other details of the model implementation such as regularisation and network depth will be discussed in the next few sections.

4.3.3 Built-in invariance

A fundamental problem of computer vision is learning to separate the properties of the image from the inherent properties of the object itself. In practical terms, this means that the object detector has to be invariant to shifts in viewpoint, lighting, noise, and other conditions caused by the properties and placement of the camera and by the environment of the object, rather than the object itself. We have shown that using deep learning techniques we can acquire most of this invariance automatically from the training data, given that the training data is sufficiently diverse as we explained in chapter 3. This way, it is up to the data to show which invariance types are useful to learn. The developer of the system does not have to take each one into consideration separately. However, while learning the invariance is a useful property, it has to be balanced with practical considerations, such as efficiency. Convolutional networks differ somewhat from "blank slate" deep learning in that they force translation invariance to hold as a structural property of the network. Pooling and stride operations which are a big part of the CNN architecture where in our model we have used the max pooling and strides size of 4,4 in most of the layers in the final AlexnetV2. These operations cause invariance to small changes in pixel-level information while keeping the network down to a manageable size.

Invariance assumptions also can be made by the region proposal methods. Selective Search and Edge Boxes evaluate the objectness based on hand-crafted techniques. Faster R-CNN and other integrated methods sidestep this by learning to generate the regions. Later discussion and potential improvements will be mentioned in the section further work.

4.3.4 Network depth

Since deep networks have been responsible for much of the improvement in image processing/classification given any task, it is relevant to ask whether the networks could be made deeper still. For **AlexNet** networks (our base model), it has been experimentally found that, for a given problem, there is a certain optimal number of layers, after which training and test error start to increase. This is called the degradation problem. The reasons for this behavior are currently not wholly known. So the only way to manage this is in an empirical way where you try and see the results and it is exactly what we did. In the example in **AlexNetV2**, we have removed 4 layers in the dense part of the network and 3 convolutional layers. In the past, the back-propagation algorithm suffered from vanishing gradients, but this problem was mitigated by replacing sigmoidal activation functions with (Relu) rectified linear units as in our case we used them with Tanh (Hyperbolic function).

4.3.5 Batch size Normalisation Impact on Generalisation

We have trained our last version of the model **AlexNetV2** using stochastic gradient descent (SGD) variant called **adam**. This method (and its variants) operate in a small-batch regime wherein a fraction of the training data, say 32–512 data points, is sampled to compute an approximation to the gradient.

It has been observed in practice that when using a larger batch there is a degradation in the quality of the model, as measured by its ability to generalize as it was detailed in [16] paper (on large-batch training for deep learning: generalization gap and sharp minima). They investigate the cause for this generalization drop in the large-batch regime and present numerical evidence that supports the view that large-batch methods tend to converge to sharp minimizers of the training and testing functions and as is well known, sharp minima lead to poorer generalization. In contrast, small-batch methods consistently converge to flat minimizers, following up on their result and after we have evaluated our model on the testing set we decided to change our batch size from 32 to 16 and then to 8 in the final architecture and it did effectively improve the generalization gap by 3 percent.

Deep Learning has emerged as one of the cornerstones of large-scale machine learning. The problem of training these networks is one of non-convex optimization. Mathematically, this can be represented as 4.14:

$$\min_{x \in \mathbb{R}^n} \quad f(x) := \frac{1}{M} \sum_{i=1}^{M} f_i(x),$$

Figure 4.14: Non-convex optimization Mathematical representation

where f(x) is a loss function for data point I 1, 2, · · · , M which captures the deviation of the model prediction from the data, and x is the vector of weights being optimized. The process of optimizing this function is also called training of the network. Stochastic Gradient Descent (SGD) and its variants are often used for training deep networks and that is why we have decided to go with it. These kinds of methods minimize the objective function f by iteratively taking steps of the form 4.15:

$$x_{k+1} = x_k - \alpha_k \left(\frac{1}{|B_k|} \sum_{i \in B_k} \nabla f_i(x_k) \right),$$

Figure 4.15: Equation of optimizer stepping

where Bk 1,2, · · ·,M is the batch sampled from the dataset and k is the step size an iteration k. These methods can be interpreted as gradient descent using noisy gradients, which and are often referred to as mini-batch gradients with batch size |Bk|. SGD and its variants (like adam) are employed in a small-batch regime, where |Bk | M and typically |Bk | 32, 64, · · · , 512. These configurations have been successfully used in practice for a large number of applications not just ours. Many theoretical properties of these methods are known. These include guarantees of (a) convergence to minimizers of strongly-convex functions and stationary points for non-convex functions), (b) saddle-point avoidance, and (c) robustness to input data. Stochastic gradient and it's variant methods have, however, a major drawback: owing to the sequential nature of the iteration and small batch sizes, there is a limited avenue for parallelization. While some efforts have been made to parallelize SGD for Deep Learning, the speed-ups and scalability obtained are often limited by the small batch sizes. One natural avenue for improving parallelism is to increase the batch size |Bk|. This increases the amount of computation per iteration, which can be effectively distributed. However, this leads to a loss in generalization performance. In other words, the performance of the model on testing data sets is often worse when trained with large-batch methods as compared to small-batch methods. In their experiments, Jorge and Mikhail have found a drop in generalization (also called generalization gap) to be as high as 5 percent even for smaller networks [20].

4.3.6 Regularization of Deep Learning model

Regularization is a central problem in machine learning is how to make an algorithm that will perform well not just on the training data, but also on new inputs. Many strategies used in machine learning are explicitly designed to reduce the test error, possibly at the expense of increased training error. These strategies are known collectively as regularization. A great many forms of regularization are available to the deep learning practitioner whether in theory or some of them are embedded in implementation frameworks such as TensorFlow. In fact, developing more effective regularization strategies has been one of the major research efforts in the field. In practice, an overly complex model family does not necessarily include the target function or the true data-generating process, or even a close approximation of either. We rarely have access to the true data-generating process so we can never know for sure if the model family being estimated includes the generating process or not. Most applications of deep learning algorithms, however, are to domains where the true data-generating process is almost certainly outside the model family. Deep learning algorithms are typically applied to extremely complicated domains such as our case images. For which the true generation process essentially involves simulating the entire universe. To some extent, we are always trying to fit a square peg (the data-generating process) into a round hole (our model family). What this means is that controlling the complexity of the model is not a simple matter of finding the model of the right size, with the right number of parameters. Instead, we might find—and indeed in practical deep learning scenarios, we almost always do find—that the best fitting model (in the sense of minimizing generalization error) is a large model that has been regularized appropriately.

In few next paragraphs, we will briefly discuss some of the regularization

techniques used across the field and that we took into consideration before choosing the one that we used.

Parameter Norm Penalties

Regularization has been used for decades prior to the advent of deep learning. Linear models such as linear regression and logistic regression allow simple, straightforward, and effective regularization strategies. Many regularization approaches are based on limiting the capacity of models, such as neural networks, linear regression, or logistic regression, by adding a parameter norm penalty to the objective function J. We denote the regularized objective function by J (Figure 4.16):

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta}),$$

Figure 4.16: Parameter Norm Penalties for an objective function

where [0,) is a hyperparameter that weighs the relative contribution of the norm penalty term, , relative to the standard objective function J. Setting alpha to 0 results in no regularization. Larger values of alpha correspond to more regularization. When our training algorithm minimizes the regularized objective function J it will decrease both the original objective J on the training data and some measure of the size of the parameters (or some subset of the parameters). Different choices for the parameter norm omega can result in different solutions being preferred. In this section, we discuss the effects of the various norms when used as penalties on the model parameters, however, this method performed quite poorly compared to the final network regulator that we have used (dropout) but it is worth it to try all the methods since it is an empirical field.

L1 and L2 Parameter Regularization

In terms of layers, weight regularization w have also considered the L1 and L2 methods. The L2 parameter norm penalty is commonly known as weight decay. This regularization strategy drives the weights closer to the origin1by adding a regularization term ridge regression, where the RSS is modified by adding the shrinkage quantity. Now, the coefficients are estimated by minimizing this function. Here, is the tuning parameter that decides how much we want to penalize the flexibility of our model. The increase in flexibility of a model is represented by an increase in its coefficients, and if we want to minimize the loss function, then these coefficients need to be small. This is how the Ridge regression technique prevents coefficients from rising too high. Also, notice that we shrink the estimated association of each variable with the response, except the intercept 0, This intercept is a measure of the mean value of the response when xi1 = xi2 = . . . = xip = 0. its term in figure 4.17:

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}\beta_j^2=\mathrm{RSS}+\lambda\sum_{j=1}^{p}\beta_j^2$$

Figure 4.17: L2 parameter norm penalty equation

Lasso is another variation that we took into consideration, in which the above function is minimized. It's clear that this variation differs from ridge regression only in penalizing the high coefficients. It uses |j|(modulus)instead of squares of , as its penalty. In statistics, this is known as the L1 norm.

Bagging and Other Ensemble Methods

Bagging(short for bootstrap aggregating) is a technique for reducing generalization error by combining several models. The idea is to train several different

models separately, then have all the models vote on the output for test examples. This is an example of a general strategy in machine learning called model averaging. Techniques employing this strategy are known as ensemble methods. The reason those model averaging works is that different models will usually not make all the same errors on the test set. Different ensemble methods construct the ensemble of models in different ways. For example, each member of the ensemble could be formed by training a completely different kind of model using a different algorithm or objective function. Bagging is a method that allows the same kind of model, training algorithm, and the objective function to be reused several times. Specifically, bagging involves constructing k different datasets. Each dataset has the same number of examples as the original dataset, but each dataset is constructed by sampling with replacement from the original dataset. This means that, with high probability, each dataset is missing some of the examples from the original dataset and contains several duplicate examples. The model i is then trained on dataset i. The differences between examples are included in each dataset result in differences between the trained models, however when we combined the SGD algorithm with the dropout (which we will detail in the next section) it resulted in the same results because the SGD do exactly the same thing but in a different way. It divides the data into mini-batches combined with the dropout that isolates some part of the network at a random time during the training by dropping some of the neurons [3].

Dropout

Dropout is one of many choices that we have used in our experiment as it was detailed in the chapter 3 in the iterations second, third, and forth where we have added potential dropout neurons/layers throughout all the network, this method provides a computationally inexpensive but powerful method of regularizing a

broad family of models. To a first approximation, dropout can be thought of as a method of making bagging practical for ensembles of very many large neural networks. Bagging involves training multiple models and evaluating multiple models on each test example. This seems impractical when each model is a large neural network, since training and evaluating such networks is costly in terms of run-time and memory. It is common to use ensembles of five to ten neural networks—Szegedy et al. (2014a) used six to win the ILSVRC—but more than this rapidly becomes unwieldy. Dropout provides an inexpensive approximation to training and evaluating a bagged ensemble of exponentially many neural networks. Specifically, dropout trains the ensemble consisting of all sub-networks that can be formed by removing non-output units from an underlying base network. In most modern neural networks, based on a series of affine transformations and non-linearities, we can effectively remove a unit from a network by multiplying its output value by zero. This procedure requires some light modification for models such as radial basis function networks, which take the difference between the unit's state and some reference value. The figure **??** below shows dropout trains an ensemble consisting of all sub-networks [3].

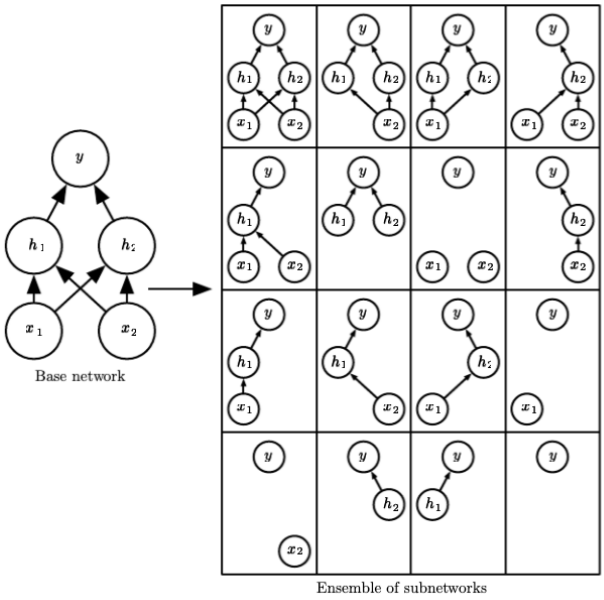


Figure 4.18: Dropout trains an ensemble consisting of all subnetworks

Many other regularization techniques exist out there in the field from which we can name: Dataset Augmentation, Adversarial training, Noise Robustness, Semi-Supervised Learning, transfer learning ..., we have chosen these specific ones that we took into our consideration when we built our model even though we dropped some of them from the final architecture, and then we ended up using the dropout method combined with SGD algorithm.

4.3.7 Determining Whether to Gather More Data

After the first end-to-end system is established, it is time to measure the performance of the algorithm and determine how to improve it. Many machine learning novices are tempted to make improvements by trying out many different algorithms. However, it is often much better to gather more data than to improve the learning algorithm [3]. How can we decide whether to gather

more data? First, we determine whether the performance on the training set is acceptable. If performance on the training set is poor, the learning algorithm is not using the training data that is already available, so there is no reason to gather more data. Instead, it is recommended to try to increase the size of the model by adding more layers or adding more hidden units to each layer to extract more hidden features. Also, it is recommended to try to improve the learning algorithm, for example by tuning the learning rate hyper-parameter, however in our case by using the adam optimizer the learning rate is adjusted during the training automatically. If large models and carefully tuned optimization algorithms do not work well, then the problem might be the quality of the training data. The data may be too noisy or may not include the right inputs needed to predict the desired outputs. This suggests starting over, collecting cleaner data, or collecting a richer set of features. If the performance on the training set is acceptable, then measure the performance on a test set. If the performance on the test set is also acceptable, then there is nothing left to be done. If test set performance is much worse than training set performance, then gathering more data is one of the most effective solutions. When deciding whether to gather more data, it is also necessary to decide how much to gather. It is helpful to plot curves showing the relationship between training set size and generalization error. By extrapolating such curves, one can predict how much additional training data would be needed to achieve a certain level of performance. Usually, adding a small fraction of the total number of examples will not have a noticeable impact on generalization error. It is therefore recommended to experiment with training set sizes on a logarithmic scale, for example doubling the number of examples between consecutive experiments [3].

If gathering much more data is not feasible, the only other way to improve

generalization error is to improve the learning algorithm itself. This becomes the domain of research and not the domain of advice for applied practitioners [3].

Chapter 5

Deployment (Software As service)

In this chapter, we will be discussing the post-development phase of our project which is the deployment of the model as SAS (Software as service). It can be seen as an explanation of the technologies used to achieve this. Note that some part of the architecture explained later in this part is still under development or Beta Version.

5.1 Serving of the model

5.1.1 Tensorflow Serving

Tensorflow Serving is an open-source ML model serving project by Google. In Google's own words, "Tensorflow Serving is a flexible, high-performance serving system for machine learning models, designed for production environments. It makes it easy to deploy new algorithms and experiments while keeping the same server architecture and APIs. Tensorflow Serving provides out-of-the-box

integration with Tensorflow models, but can be easily extended to serve other types of models and data." figure 5.1 shows the architectures of tensforflow serving framework

There are a number of ML model serving platforms in the market right now. We chose Tensorflow Serving because of these three reasons, ordered by priority:

- **Highly performant** . It has proven performance handling tens of millions of inferences per second at Google according to their website.

- **Highly available**. It has a model versioning system to make sure there is always a healthy version being served while loading a new version into its memory.

- **Actively** maintained by the developer community and backed by Google Even though, by default, Tensorflow Serving only supports models built with Tensorflow, this is not a constraint, though, because Grab is actively moving toward using Tensorflow [21].

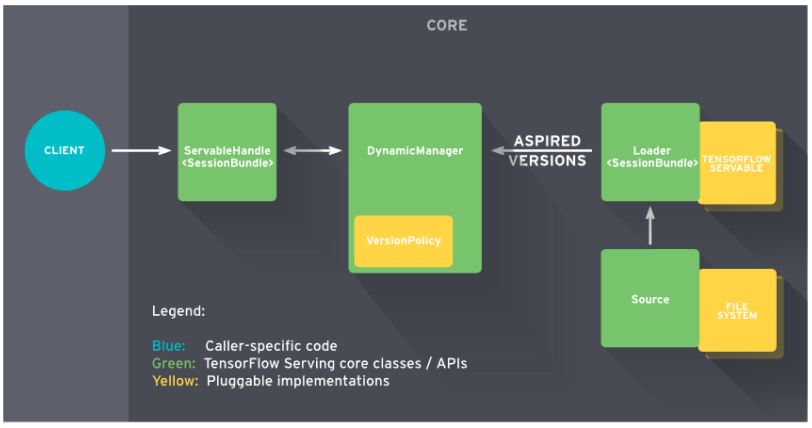


Figure 5.1: Architectures of Tensorflow Serving Framework [21].

5.1.2 Docker and google cloud

Docker is a set of platform-as-a-service (PaaS) products that use operating-system-level virtualization to deliver software in packages called containers. Containers are isolated from each other and group their own software, libraries, and configuration files. They can communicate with each other through well-defined channels. All containers are run by a single operating system kernel and therefore use fewer resources than virtual machines [22]. The docker is a virtualization alternative in which the host machine's kernel is shared with the virtualized machine or the software in operation, so a developer can add to his software the possibility of taking the libraries and other dependencies of his program together with the software with less performance loss than hardware virtualization of a complete server. Thus, the docker makes operations on infrastructure such as web services more interchangeable, efficient, and flexible.

5.1.3 Make First HTTP Request

Using the two tools explained in the previous two sections we have built a docker image that contains a servebale Tensorflow model with the H5 Format that can be served in a container exposing a Restful API as well as GRPC endpoint to make the use of HTTP 2. The Figure 5.2 shows how to pull the image from Gitlab Registry where the image is stored.
Link to the Repository :`https://gitlab.com/master-thesis8/model-tensforflow`
(it may require the project admin to give you access)

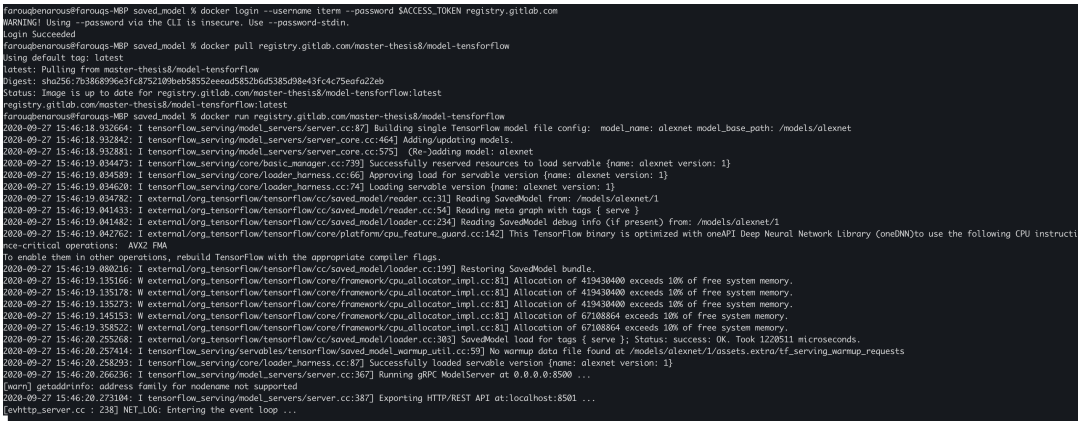


Figure 5.2: Terminal explanation of the hhtp request

```
#This step may require credentials from the owner of the Repo
docker login --username iterm --password $ACCESS_TOKEN

# This step take about 5 minutes since the model has more than 1 gb size
docker pull registry.gitlab.com/master-thesis8/model-tensforflow

# Running the image
docker run registry.gitlab.com/bytepitch/imagine20x/database

# Check if the Container is running
docker ps
```

After making these steps the model will be served at the port 8500-8501/tcp and ready to use. After making these steps the model will be served at the port 8500-8501/tcp and ready to use. An image must be sent to the model in form of a NumPy array by the size 224 224 3. You can use our image processing API in order to process your image the next section would be an explanation of how to do that.

5.2 Back-end for the model

5.2.1 Overview Architecture

After We've Served the Tensorflow Model in a docker container the next step is to create a back end to be served as software to the end client the figure below shows the overview architecture of the different pieces of the software.

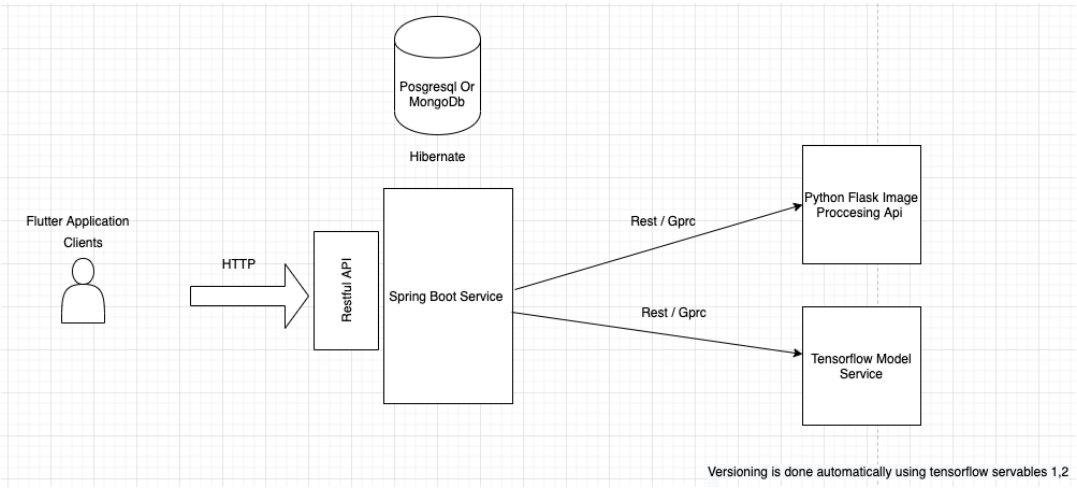


Figure 5.3: Overview Architecture of the software

The Tensorflow Model Piece has already been explained in the chapter 4, therefore in the next section we will be detailing other components shown in the figure 5.3.

Image Processing API

This part is the Web Service of the image processing techniques that we have developed and explained in the Data Collection and pre-processing part.

1. Protobuf vs. JSON : One of the biggest differences between REST and gRPC is the format of the payload. REST messages typically contain

JSON. This is not a strict requirement, and in theory you can send anything as a response, but in practice the whole REST ecosystem—including tooling, best practices, and tutorials—is focused on JSON. It is safe to say that, with very few exceptions, REST APIs accept and return JSON. gRPC, on the other hand, accepts and returns Protobuf messages, from a performance point of view, Protobuf is a very efficient and packed format. JSON, on the other hand, is a textual format. You can compress JSON, but then you lose the benefit of a textual format that you can easily expect [23].

2. HTTP/2 vs. HTTP 1.1 : Let's compare the transfer protocols that REST and gRPC use. REST, as mentioned earlier, depends heavily on HTTP (usually HTTP 1.1) and the request-response model. On the other hand, gRPC uses the newer HTTP/2 protocol. There are several problems that plague HTTP 1.1 that HTTP/2 fixes. Here are the major ones. HTTP 1.1 is too big and complicated. The growth of page size and the number of objects, latency issues, head of line blocking. However, the major improvement of HTTP/2 is that it uses multiplexed streams. A single HTTP/2 TCP connection can support many bidirectional streams. These streams can be interleaved (no queuing), and multiple requests can be sent at the same time without a need to establish new TCP connections for each one. In addition, servers can now push notifications to clients via the established connection (HTTP/2 push) [23].

3. REST supports only the request-response model available in HTTP 1.x. But gRPC takes full advantage of the capabilities of HTTP/2 and lets you stream information constantly. There are several types of streaming: Bidirectional Streaming ,Client-Side Streaming , Server-Side Streaming

[23] .

4. Strong Typing vs. Serialization : The REST paradigm doesn't mandate any structure for the exchanged payload. It is typically JSON. Consumers don't have a formal mechanism to coordinate the format of requests and responses. The JSON must be serialized and converted into the target programming language both on the server side and client side. The serialization is another step in the chain that introduces the possibility of errors as well as performance overhead. The gRPC service contract has strongly typed messages that are converted automatically from their Protobuf representation to your programming language of choice both on the server and on the client. JSON, on the other hand, is theoretically more flexible because you can send dynamic data and don't have to adhere to a rigid structure [23].

5. Support for gRPC in the browser is not as mature. Today, gRPC is used primarily for internal services which are not exposed directly to the world. If you want to consume a gRPC service from a web application or from a language not supported by gRPC then gRPC offers a REST API gateway to expose your service. The gRPC gateway plugin generates a full-fledged REST API server with a reverse proxy and Swagger documentation. With this approach, you do lose most of the benefits of gRPC, but if you need to provide access to an existing service, you can do so without implementing your service twice [23].

Following up on the comparison of the two techniques of serving resources we have decided to make our system oriented toward GRPC support therefore our system will use GRPC as the main Standard, but without negligent the Rest endpoint because these APIs can also be served as a third party for other

services/client. Since our image processing was implemented using Python and OpenCV we have decided to go with python to serve these functions (make things less complicated) FLASK (python web framework) was chosen in order to do that, Link to the Repository :`https://gitlab.com/master-thesis8/image-processing-api` (it may require to the project admin to give you access). The API exposes an endpoint noted by /almonds/features the endpoint has 2 verbs one for POST and one GET, The POST is used to extract features of an image sent in the body and it returns the corresponding NumPy array to be used later on for the classification. The Project is running on docker as well exposing two ports 50051 for GRPC and 3001 Restful, of course in order to access the Grpc you will be needing the protocol buffer service definition further details about this would be in the Readme of the Repository. The Repository contains a docker-compose to run both containers together. Here are steps on how to run then:

```
 # enter the Repository
cd path/to/repo

# build the images
docker-compose build

# Run the docker containers
docker-compose up - d

# check if everything is running
docker-compose ps
```

After Running the containers you can test then by sending images to the exposed end-points via Postman or BloopRpc

Spring Boot and Postgress as Main Service

The final step of the back-end Development is to Combine the two microservices that we have built and detailed in the previous sections. We are building a spring boot web application with a Postgress database to store some of the user information for later use. This part will have these main features: Handel User Authentication and Authorization

1. Handel User Authentication and Authorisation
2. When it receives a request firstly it calls the image processing API using GRPC to get the NumPy array after applying the different masks And then it calls the Tensorflow Model according to the image git (almond, olives ...) after it gets the result from the TensorFlow model it stores all these operations in the PostgreSQL and then it serializes the response in JSON and sends it back the user.
3. Store in the database the previous performed User Request on the API
4. Store the user feedback on the classification results
5. Notify the Pipeline for Tensorflow model after a specific threshold to trigger the new training of the model with the new data and deploy the new version if it has better accuracy

End Client

The client-side of the software is gonna be a Flutter mobile application, for a bit of context about flutter Flutter is an open-source user interface development kit (UI toolkit), created by Google, which allows the creation of natively compiled

applications. Currently, it can compile for Android, iOS, Windows, Mac, Linux, Google Fuchsia, and the Web. Flutter is becoming on of the leading frameworks for mobile app development. This part is still under development in the design phase, where the app is gonna have two layouts one of login/signup after the user is signing it will show a layout with a list of the previous operation performed on the API (previous classifications) as well as a button that will activate the camera and therefore taking a picture to send, perform HTTP request to API and then get the classification results back.

Chapter 6

Conclusion

6.1 Reviews and usefulness of the solution

AI and Deep learning have application in countless areas, any field can eventually use it to solve problems that once we thought they are unsolvable. The model that we have developed in this thesis is just a drop in the ocean on how Deep Learning can help solve problems, optimize the existing solution and develop tools that will help change any industry. The biggest advantage of the presented algorithm is its high precision. The developed classifier is able to detect many types of almonds (16 in our case of study and it can be increased) with a high accuracy that reached 94 percent in the last iteration of the development. We can also emphasize on the solution low-cost and on the model scalability since it can scale to any other case of studies using methods like transfer learning. Moreover our solution can be easily used and adapted to work in different devices.

This kind of solutions can be used in different contexts including in industry real problems, such as:

1. Supply chain optimization – less waste and more transparency As long

as food manufacturers are concerned with food safety regulations, they need to appear more transparent about the path of food in the supply chain. Here, AI in food manufacturing helps to monitor every stage of this process — it makes price and inventory management predictions and tracks the path of goods from where they are grown to the place where consumers receive it, ensuring transparency. A solution such as Symphony Retail AI enables us to estimate the demand for transportation, pricing, and inventory to avoid getting an abundance of goods that end up wasted.

2. Sorting food: optical sorting solutions Previously, a manufacturer had to hire many people to perform the monotonous and routine actions linked to food selection. Now, instead of manually sorting large amounts of food by size and shape (so that it can be canned or bagged), you can use AI-based solutions to easily recognize which plants should be potato chips and which are better to use for french fries. Vegetables of an inappropriate color will also be sorted out by the same system, decreasing the chance that they are discarded by buyers. Food Sorters and Peelers developed by TORMA show better processing capacity and availability, which increased food quality and safety. This is achieved by using core sensor technologies and a camera that recognizes material based on color, biological characteristics, and shape (length, width, diameter); the camera has an adaptive spectrum that is well suited for optical food sorting.

3. Predictive maintenance, remote monitoring, and condition monitoring It is obvious that manufacturing a lot of goods demands large, complicated, and intricately constructed mechanisms. The maintenance of such machines can be rather costly without predictive maintenance – figuring out the time-to-repair and cost-to-repair indicators through categorizing issues

and making predictive alerts. Timely repairs can save up to 50 percent maintenance time and reduce the costs needed for it by almost 10. To perform remote monitoring on complicated mechanisms, you can make a Digital Twin of a machine that will show you the performance data on parameters and manufacturing processes and boost the throughput. Machine Learning also allows the identifications of factors that affect the quality of the manufacturing process with Root Cause Analysis (eliminating the problem at its very source). With condition monitoring, you can monitor the equipment's health in real-time to reach high overall equipment effectiveness (OEE) [24].

4. Matching customer tastes with your business strategy, the idea is to use historical data to know better each client and their needs. Providing service and goods that match completely the taste and the needs of each client will improve the business.

6.2 Further work

As a final conclusion for the thesis and for the work that we have done, I wanted to present some of the ideas/ features that can be developed based on our work. As I mentioned earlier, our work is a drop in the sea from what deep learning can solve in all the fields and all the world, so in this section, you will see Further work and extents of our thesis project.

6.2.1 Further work on the model

The convolutional model and the image processing that we built can be used as a starter model for other similar issues that IPB agricultural Lab faces. For example, one of the ideas is to apply it on olives classification or even in the same

use case (almonds). In this case, we can extend it to more classes of almonds or we can consider classifying by quality of the almond, by conservation time and also it is possible to infer the thickness of the shell to know what type of machine should be used to peel them.

About image processing improvements as explained in the classification part all the methods that we have used are kind of static methodologies. All the masks and filters are generic algorithms, defined by default in OpenCv 2.0, a further work can be conducted here making these parameters learnable. For example, where we used the Canny thresh-hold or the blur kernel size, these parameters can be trained toward one dataset and it will have a better image segmentation results (both accuracy and generalization).

A general improvement for the Cnn model and the image processing together, is to introduce a region proposal techniques. In our work, we have assumed that the picture that we feed to the model has only one almond and we segment the image based on that, wherein the case of using a region proposal techniques such as Mask Cnn, Fast R-Cnn, Faster Rcnn the extraction of the number of almonds, separation of almond from the background would work for images with multiple almonds in it.

When training the CNN, GPU implementation is necessary. Training the CNN with merely a CPU implementation can take months to complete for the complete GS1US product category dataset. It is logical to conclude that fine-tuning the system during this process is not feasible in that time setting. Scalability does not seem to be an issue with the implementation of GPU computation, and furthermore, the cloud-based computation can be implemented to decrease training time even more, when training very large datasets.

6.2.2 Further development on the software

As the model can scale for further features the software around should be able to scale as well and should be able to scale with all the potential features that we have discussed in the previous section. In order to do that the back end that we built is totally loosely coupled where the 3 different components of it can be developed/deployed separately with no issue at all. For example, the image processing API has an EndPoint for almonds under /almonds/features and we can as well add olives in the same API by just adding it to the route as /olives/features. The load balancer in front of it so you can add more instances of the container and treat the requests in the different instances. The Tensorflow Serving Component also is highly scalable where TF team recently added the multiple model deployment per container where you can just give the container a JSON models config as the following code shows instead of just one model:

```
# json config of the different moldes
model_config_list: {
config: {
name: "model1",
base_path: "/tmp/model",
model_platform: "tensorflow"
},
config: {
name: "model2",
base_path: "/tmp/model2",
model_platform: "tensorflow"
}
    }
```

We might need to add a Load balancer here as well, it would be as easy as the first component since all the APIs are based on restful conventions.
The third component which is the liaison between the two is built using Spring Boot and Postgresql which are battle-tested matured technologies and we can easily develop new features in using them. We might just need to add caching layers using Redis to speed-Up requests processing times.

Hopefully, the ideas in this dissertation will serve as a useful foundation for the design and architecture of future works, and they will encourage further research on Deep learning in agricultural-related areas.

Bibliography

[1] B. E. Rasool Khodabakhshian, M. Khojastehpour, and M. R. Golzarian, "Combination of conventional imaging and spectroscopy methods for food quality evaluation", *Conference: WCSE 2014At: Dubai, UAE*, 2014. [Online]. Available: https://www.researchgate.net/publication/264991916_Combination_of_conventional_imaging_and_spectroscopy_methods_for_food_quality_evaluation.

[2] B. R. Haohan Wang, "On the origin of deep learning", *conference Neural and Evolutionary Computing (cs.NE*, 2017. [Online]. Available: https://arxiv.org/abs/1702.07800.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[4] J. Schmidhube, "Deep learning: Our miraculous year 1990-1991", 2019. [Online]. Available: http://people.idsia.ch/~juergen/deep-learning-miraculous-year-1990-1991.html.

[5] E. T. Delila Halac Emir Sokic, "Almonds classification using supervised learning", *XXVI International Conference on Information, Communication and Automation Technologies (ICAT)*, 2017. [Online]. Available: https://ieeexplore.ieee.org/document/8171603/.

[6] P. Domingos, *The master Algorithm By Pedro Domingos*. Basic Books, 2015.

[7] *Almond produtction should double in a decade in portugal*, https://www.sisab.pt/noticias/producao-de-amendoa-devera-duplicar-numa-decada-em-portugal/?lang=en, 2018.

[8] H. Adel, A. mahmoudi, and S. Khalesi, "Detection of walnut varieties using impact acoustics and artificial neural networks (anns)", *Modern Applied Science*, 2011. [Online]. Available: https://www.researchgate.net/publication/267231434_Detection_of_Walnut_Varieties_Using_Impact_Acoustics_and_Artificial_Neural_Networks_ANNs.

[9] S. Chatterjee, *What is feature extraction? feature extraction in image processing*, https://www.mygreatlearning.com/blog/feature-extraction-in-image-processing/, Oct. 2020.

[10] O. Stenroos, "Object detection from images using convolutional neural networks", *semanticscholar*, 2017. [Online]. Available: https://www.semanticscholar.org/paper/Object-detection-from-images-using-convolutional-Stenroos/a6ee78ea9c68d99d6545227fed925a721337bb16.

[11] E. Becker, *Image processing with python*, https://orcid.org/0000-0002-6832-0233, 2020.

[12] J. CANNY, "A computational approach to edge detection", *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, 1986. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.420.3300&rep=rep1&type=pdf.

[13] J. Serra, *Image Analysis and Mathematical Morphology*. Reading, Massachusetts: Addison-Wesley, 1983.

[14] E. Technologies, *Fundamentals of image contours*, https://medium.com/@evergreenllc2020/fundamentals-of-image-contours-3598a9bcc595, 2020.

[15] kang atul., *Suzuki contour algorithm opencv*, https://theailearner.com/tag/suzuki-contour-algorithm-opencv/, 2019.

[16] G. Bradski and A. Kaehler, *Learning OpenCV*. Beijing · Cambridge · Farnham · Köln · Sebastopol · Taipei · Tokyo: O'REILLY, 2006.

[17] A. Géron, *Hands-On Machine Learning With Scikit-Learn, Keras, And Tensorflow Concepts, Tools, And Techniques To Build Intelligent Systems*. USA: O'REILLY, 2019.

[18] K. Dubovikov, *Pytorch vs tensorflow — spotting the difference*, https://towardsdatascience.com/pytorch-vs-tensorflow-spotting-the-difference-25c75777377b, 2017.

[19] A. K. Ilya Sutskever Geoffrey E. Hinton, "Imagenet classification with deep convolutional neural networks", *Communications of the ACM, 2017*, 2012. [Online]. Available: https://papers.nips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[20] P. T. P. T. Nitish Shirish Keskar Dheevatsa Mudigere Jorge Nocedal Mikhail Smelyanskiy, "On large-batch training for deep learning: Generalization gap and sharp minima", *conference paper at ICLR 2017*, 2017. [Online]. Available: https://arxiv.org/abs/1609.04836.

[21] M. andAshish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh

Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: `https://www.tensorflow.org/`.

[22] B. GOLUB, *Who sold gluster to red hat, now running dotcloud*, `https://icloud.pe/blog/ben-golub-who-sold-gluster-to-red-hat-now-running-dotcloud/`, 2013.

[23] G. Sayfan, *Rest vs. grpc: Battle of the apis*, `https://code.tutsplus.com/tutorials/rest-vs-grpc-battle-of-the-apis--cms-30711`, 2018.

[24] O. Kovalenko, *Machine learning and ai in food industry: Solutions and potential*, `https://spd.group/machine-learning/machine-learning-and-ai-in-food-industry/`, 2020.