Alexandru Ioan Cuza
University of Iași

Faculty of
Computer Science

Fruit Dataset. Fruit Classification.

~ Bachelor Thesis ~

Session of July

2021

Proposed by

Minuț Mihai-Dimitrie

Scientific Coordinator

Prof. Dr. Adrian Iftene

UNIVERSITY ”ALEXANDRU IOAN CUZA” OF IASI
FACULTY OF COMPUTER SCIENCE

Bachelor Thesis

Fruit Dataset.

Fruit Classification.

Session of July
2021

Proposed by
Minuț Mihai-Dimitrie

Scientific Coordinator
Prof. Dr. Adrian Iftene

Advised,

Paper Directive

Title, Surname and Last name *Prof. Dr. Adrian Iftene*

Date 01-07-2021 *Signature:*

DECLARATION

regarding the originality of the content of the bachelor thesis

I, Minuț Mihai-Dimitrie, with domicile in the Piatra Neamț city, Neamț county born on 10.06.1998, identified by CNP 1981006070054, graduate of "Alexandru Ioan Cuza" University of Iași, Faculty of Computer Science specialization Computer Science, promotion 2021, declare on my own responsibility, knowing the consequences of false statements in the sense of art. 326 of the New Penal Code and the provisions of National Education Laws nr. 1/2011 art. 143 al. 4 and 5 regarding plagiarism, that my bachelor thesis entitled: "Fruit Dataset. Fruit Classification." developed under the guidance of Prof. Dr. Adrian Iftene, which I am going to support before the committee, is original, belongs to me and I assume its entire content.

I, as well, declare that I agree of my bachelor thesis being verified through any legal method for confirming its authenticity, inclusively allowing its contents to be stored in a database for this purpose.

I have taken note of the fact that it is forbidden to commercialize scientific papers in order to facilitate the falsification by the buyer of the authorship of a bachelor's thesis, diploma or dissertation and in this regard, I declare on my own responsibility that this paper was not copied and represents the result of the research that I have done.

Today's Date *Student's Signature*

01-07-2021

DECLARATION

REGARDING THE ORIGINALITY AND RESPECT OF COPYRIGHTS

I hereby declare that the Bachelor Thesis entitled ”Fruit Dataset. Fruit Classification.” is written by me and has never been presented to another college or higher education institution from the country or from abroad. I also declare that all the sources that have been used, including the ones taken from the Internet, are indicated in the paper, in compliance with the rules to avoid plagiarism:

- all the text fragments that are reproduced exactly, including the ones translated from one language to another, are written inside quotation marks and have attached a reference of the original source;

- the reformulation in one’s own words of texts written by other authors has attached a precise reference to the source;

- source code, images etc. taken from open-source projects or other sources are used in accord with copyrights and have attached precise references;

- the summary of the ideas of other authors specifies the precise reference to the original text.

Graduate Minuț Mihai-Dimitrie

Iași, July 2021

DECLARATION OF CONSENT

I hereby agree that the Bachelor Thesis entitled ”Fruit Dataset. Fruit Classification.”, the source code and all the other content (graphical data, test data, etc.) that come together with this thesis to be used within the Faculty of Computer Science.

I also agree that the Faculty of Computer Science from the University ”Alexandru Ioan Cuza” from Iași can use, modify, reproduce and distribute for non-commercial purposes the computer programs, source and executable format, made by me in my Bachelor Thesis.

Iași, July 2021

Graduate Minuț Mihai-Dimitrie

DECLARATION OF COPYRIGHT REGARDING THE DATASET

I hereby declare that I do not own any of the images used within the image dataset and that I did not and nor will I use any of the images in any form of commercial purpose. All of the images have their own respective authors that made them available on public domain.

I also declare that the only actions I did upon the images were collecting from the public domain and grouping them into specific categories, the result being the creation of the entiry of the dataset.

I also declare that I do not own any part of the dataset and that the dataset is entirely build under public domain dedication (CC0 1.0).

Graduate Minuț Mihai-Dimitrie

Iași, July 2021

Summary

1. Introduction

1.1. Motivation

Even before beginning my studies at a computer science faculty I was amazed by the evolution of autonomous robots and artificial intelligence. My studies have shown me what the back end of such applications generally consist of and made me want to understand and research more.

The field of AI has suffered a lot from evolution and is still under lots of development which further proves the utility of additional work and research in it. Deep learning and convolutional neural networks are a prime example of the above statement **[1]** due to the popularity they have in the field of computer vision.

A popular subject in the city, in which my faculty resides in, is the Botanical Garden and as a consequence is a great target for application development. This has made it a great target for many applications along the timeline.

As a result the target of this thesis is to elaborate an algorithm or a method to help in the development of such applications, regarding the Botanical Garden, while also maintaining a general perspective on the problems that appear along the way so the results may be useful in as many circumstances as possible.

1.2. Problem

The first part of the problem that I decided to take on is the building of a very large dataset of plants with as many specimens if not all of them. The dataset should respect the norms and regulations that other public and online datasets respect and should contain information and data of high quality.

The second part of the proposed problem is to build an algorithm that will take as input an image of a plant and will produce as output a prediction of what plant the image contains. The aim for the algorithm is to have as much accuracy as possible and to be able to be utilized in other applications that require the feature of plant recognition.

To accurately bring into light the strengths and weaknesses of the components mentioned above multiple statistics and statements, that results from them, will have to be made along the way.

The dataset and all of its usable variants as well as some of the best models, results and statistics should be made available to the public.

2. Work of Others

2.1. Introduction

First step into solving the problem is gathering knowledge within the domain of the problem, particularly image classification and android mobile applications. Analyzing existing and theoretical concepts, state of the art and different applications brings into the light what is or not feasible in these times with limited resources while also allows for discovery when it comes to popular technologies, patterns and design.

2.2. Image Classification

2.2.1. Introduction

The task of Image Classification consists of analyzing the data of an image and producing a result that represents a label, thus assigning a form of meaning to the target image. In order to make the process work, the image classified should contain only one type of label from the label pool provided by the classifying task **[2]**.

2.2.2. Artificial Intelligence

Since the first apparitions of computers people have tried to accomplish the task of making the machines think like humans, as a result the Artificial Intelligence field has been born **[3]**.

But even to this day the process of human thought is not fully defined and the consequence is that the field of AI has added the branch of computers that think rationally **[4]**. This subfield usually revolves around a rational thinking agent which is defined as something that thinks and acts, analyzes and takes action **[4]**.

The Total Turing Test, which derived from the Turing Test proposed by Alan Turing in 1950, broke the definition of an intelligent computer into a couple of problems: natural language processing, knowledge representation, automated reasoning, machine learning, computer vision and robotics **[4]**.

While the passing of the Turing Test was not considered important in the field of AI, the problems imposed by it transformed into their own sub-domains within the field of AI and continued evolving across the timeline even to this day **[4]**.

2.2.3. Computer Vision

The field of Computer Vision is concerned with how can computers extract information from image data **[6]**. The information extracted can be defined in many ways, depending on the situation, but common ones include 3D or higher dimension information, temporal information and special patterns, and the image data can be defined as time varying multi-dimensional data, examples ranging from simple image pixel data to sequences of images and higher dimension data. **[5] [6]**

2.2.4. Hardware Evolution

From the time Computer Vision established itself as a fully pledged subfield of Artificial Intelligence up until recent years, the hardware industry continued to advance at a rate comparable to the one described by the Moore's Law **[8]**. As a result, computational power has seen, since the beginning of Computer Vision, a very big and clear leap. As a consequence of the hardware evolution many of the theoretical and non-feasible algorithms from many fields have become now more practical and are seeing more usage **[9]**.

Computer Vision has also been struck by the hardware evolution, especially by the evolution of graphical cards, and has achieved new heights in terms of efficiency and accuracy through the use of newly feasible algorithms and practices such as Machine Learning, Deep Learning and Convolutional Neural Networks **[7]**.

2.2.5. Machine Learning

Machine Learning is the subfield of Artificial Intelligence that centers around algorithms that learn and adapt through training on past or ongoing experiences in order to produce better results at the task they are designed to solve. **[10]**

A model is a construct that follows a certain structure and is usually the final product of machine learning algorithms that is used thereafter for completing the task imposed by the problem. The model is composed of weights, which are adaptable by training, that are used to produce the result of some input data.

The process through which a model goes is separated in three chronologically linear stages:

-training: the model learns from experiences (training data) and modifies its weights in order to become better at solving the task;

-validation: the training stage is adapted by testing the model against problem data that is

not part of the training set in order to avoid the situation where the model overfits the training samples (learns to solve only the training scenarios and not the general problem);

-testing: the final model produced by the algorithm is tested against data that did not take part in validation or training in order to grasp the efficiency in new instances of the problem.

A machine learning algorithm tries to build a better model by feeding it the input data, the features, observing how it solves the problem instance compared to the actual correct results, the labels, and adapt the model so it can achieve a greater success in future instances.

2.2.6. Dataset

Datasets are collections of data that is presented in a certain format which usually contains multiple items that respect a general archetype when it comes to the information provided. Digital datasets can consist of collections of files, documents, values, images, tables or in general any type of data that can be stored and used by computers. **[13]**

The training data utilized by learning algorithms usually comes from already established datasets. Such collections usually follow specific architectures that make them easy and efficient to be utilized by learning algorithms.

Usually datasets built for learning algorithms are well balanced (each label has approx. as much information as the other labels), a formula for the size of unitary data is frequently used and the information itself is cleaned of the majority of noise that can harm the learning.

2.2.7. Deep Learning

Deep Learning is a subdomain of machine learning that focuses on solving the problem by transforming it into a hierarchy of concepts with each concept defined as a relation between simpler concepts and so on, thus enabling the computer to learn advanced and complicated topics by building them on basic and simple ones. **[11][Introduction]**

The principles of deep learning algorithms are based on linear algebra, probability, informational theory and numeric computation. **[11][Applied Math]**

The models used in deep learning algorithms are deep feedforward networks, also know as multilayer perceptrons. Feedforward networks are models through which information flow is unidirectional, with their structure consisting of layers that represent different mathematical functions.

The chained layers are the key elements of feedforward networks that determine the depth of the model. Each layer has different number of neurons with their own individual

weights which affect the output of the given layer. The layers can be separated in input, hidden and output layers.**[11][Deep Networks]**

The activation function is the function applied to a hidden layer's value in order to produce the hidden layer final output which will be passed to the following layer. **[11]**

Loss functions are the functions that neural networks use to represent the distance between the output of the model and the label in retrospect to certain features. Trying to minimize the gradient of a loss function in order to determine the error and improve the weights of the model is called gradient descent.**[11]**

Updating all the weights requires the calculations of errors in respect to each layer. The error of the last layer is determined by the loss and activation function but the error of the hidden layers are calculated from the last layer to the input in a cascading manner proportionate to the weights per layer, the process being called back propagation.**[11]**

Optimizers are algorithms used to adapt the learning rate of the network, increasing or decreasing it depending on the evolution of the learning process.**[11]**

2.2.8. Convolutional Neural Networks

Neural networks can be applied to any type of input data but adding prior knowledge of that data to the network improves its generalization performance. Images contain pixel values but they also contain spatial data, patterns of pixels, pixel spatial data. **[12][CNN]**

Convolutional neural networks (CNNs) solve the loss of spatial information by utilizing convolutional layers within the network's architecture. Convolutional layers are formed from a multitude of learning filters, that learn spatial patterns when trained.**[12]**

The input of a convolutional layer is composed of feature maps from the previous layer, which can come even from the raw image itself and can have a multitude of channels for color, usually between one and three. The output given by the convolutional layer is influenced not only by the input features but also by the filters size, window stride (filter distance). **[12]**

In order to reduce the noise that can appear in data because of convolution, pooling layers are introduced between convolutional layers. They take the feature maps and reduce them by a rate, given by the size of pooling, utilising a method (max. pooling, avg. pooling, etc..) for the said reduction that maps each feature map to a single pixel in the output. **[12]**

Traditional convolutional networks stack a multitude of layers of convolution and pooling, and after those, add layers of fully connected layers to be able to produce outputs that can represent certain labels. The input is in this case the raw image data.**[12][Traditional CNN]**

2.3. State of Art

2.3.1. Fruits 360 Dataset

The ”Fruits 360” **[29]** is a dataset that contains 90,380 images of 131 distinct fruits and vegetables in various stages of their life that is provided under the MIT License.

Data Source

The source of the image data was a shaft in which fruits and vegetables were planted, grown and filmed with the help of a camera and of a rotating platform. To obtain the final results a background was used for filming, namely a blank sheet of paper, and an algorithm for filling the background pixels since the background had a high variance in color.

Image Properties

Some important properties about the dataset include the size being 100 *x* 100 *x* 3 pixels for each image, the name format used for storage, each label having an approximate of 490 respective images and that different varieties of the same fruit are found under different labels.

B.1 - Apple Braeburn Images - Fruits 360 Dataset

The dataset contains high quality images **(B.1)** with the fruit occupying most of each image's pixel data.

Due to the process in which the dataset was obtained the consequence of having high degrees of similarity between the images under the same labels is present. **(B.1)**

Label Properties

The number of labels when compared to a total amount of existing distinct popular clades of fruits **[17-20]** covers between 30% and 40% of them.

While the coverage of distinct fruits seems small, the branching within each distinct species varieties is presented under different labels. A good example would be the the apple species, amounting to 13 different subspecies which also include 8 different subspecies types.

Dataset Task

The dataset is by default split into images for training and images for testing thus enabling the creation of the task of building and optimizing a machine learning model given the two sets of data.

The dataset and the task associated with it come with a research paper **[31]** regarding the process of building the dataset and the methods used to train a neural network that is specifically for fruit classification.

Besides the dataset author's submission there are many others **[29]** that tried and still try to solve the task , most of which present solutions having obtained models with an accuracy above 95% on the test set.

2.3.2. CNN Techniques - Fruits 360

The ”Fruits 360” dataset presented before comes with many papers that try to solve the task of creating a convolutional neural network with high accuracy. The author of the dataset has released a paper **[31]** in which he experiments on the dataset with different techniques for CNNs and data augmentation for deep learning.

Techniques presented in the paper include training on different types of images ranging from Grayscale, HSV and raw formats to combinations of all of those, which turned out to improve the accuracy of the model. Another remarkable feature of the network is the use of 5x5 convolutional filters on each layer.

Another paper/notebook on the same dataset comes from Rafet Can Kandar on the dataset's Kaggle page **[30]** where an entire pipeline of working with the data is presented. There can be seen the usage of dropout layers in combination with image augmentation (rotation, zooming, shearing) and how it can impact the accuracy of the model and how it can solve overfitting problems on a 3x3 convolutional network.

A notebook from the same source applied on the same dataset written by Justin Ruan **[34]** provides some insight on how to import an retrain pretrained models and also how to obtain statistics based on the model and the dataset.

One of the most popular take on the ”Fruits 360” task is that of Shivam Bansal **[35]** in which he describes multiple CNN architectures like VGG, ResNet, InceptionNet, XceptionNet and their results on the dataset. The notebook also presents the concept of transfer learning, how to do it and how it can fine tune a model.

Most of the techniques presented above are also presented in the seminar ”Understanding Convolutional Networks” by David Stutz **[12]** or the ”Deep Learning” **[11]** book but different tasks come with different answers and since the problem this thesis faces is very similar to the one presented by the ”Fruits 360” dataset, it is important to look for models and results on similar matters.

All the work papers and notebooks presented above will serve as inspiration for the development of the dataset and the following models which will be built upon it.

2.4. Conclusions

Exploring the computer science related domains and their state of the art allowed for an approximate estimation of the difficulty of the problem and for breaking the big problem into smaller ones.

First step into building the application is gathering information about plants and then building a dataset that needs to contain many labels each having associated a proportionate amount of image data. The quality of the dataset will directly affect the model and its training and the quality of the application altogether.

Utilizing the newly built dataset, a model will be trained on the image data to be able to classify the plants within images. Analyzing the recent applications that recognize and classify images gives insight that the model type which will most likely give good results and also be time efficient will be a Convolutional Neural Network. Initially the models might be pretty inefficient and non-accurate but with fine tuning and experimenting, the results are likely to improve.

From various model training sessions, be they simple or advanced with the utilization of certain techniques, various statistics will be withdrawn that will help in building conclusions about the dataset, the training and the better policies and techniques that should be used when building an efficient model.

3. Work of Mine

3.1. Dataset

3.1.1. Ideal Dataset

Analyzing the state of the art and different articles, it can be observed that successful machine learning datasets cover a big variety of labels that best represent the problem of identification presented by them **[14]**.

In terms of quality, each image should contain only useful information for bettering the model, the redundancy and noise in data being as low as possible. The quantity of images should scale with the number of labels in order to make the model better at distinguishing corner cases where there is little difference between images with different labels. In addition each label should also have approximately the same quantity of image information so there are no discrepancies in what the model learns more about **[13][15]**.

3.1.2. Plants and Fruits

Domain

In order to determine the goal of the dataset, a look within the domain of plants must be taken. Examining information that is found on trusted informational websites **[16]** about existing/discovered plants one can easily realize that the plants species follow a hierarchical structure and even specimens present in very specific family branches tend to have different varieties. The amount of leaf children of the hierarchy is very big with little differences between similar species which might make the classification hard even for the human eye.

A decision needed to be made, the term ”plants” is too broad in order to build a dataset around it. The success or feasibility of a dataset and model built around all the plants in all their stages of life, seasons, families and variations were simply not in reach within the limited resources of computational power and time that are available {**5. Hardware**}.

Divide et Impera

One solution is the separation of responsibility: multiple datasets and models, one pair for each plant part or family. Plant parts such as flowers, fruit, roots, trees etc. should each have their own built datasets and their respective models. This method reduces the difficulty of each individual problem and allows the building of more accurate and efficient models since

the number of labels decreases. With them, the number of weights and time needed for training will also decrease **[12]**.

This will change how the application will work by allowing the user to specify the family of plants or plant parts for classification thus increasing the user interaction while also reducing the time needed to produce an output.

3.1.3. Building a Dataset

3.1.3.1. Domain of Data

For an application that classifies plants and parts of them, the release order of the categories needs to be influenced by how likely the user is to access that category. Most popular or easily found plant categories need to be implemented first. From the pool of common plants or plant parts a regular encounter for humans are fruit, which are usually found through a lot of mediums: any natural habitat, at markets, or even at home. This makes them one of the better candidates for the first label to have a dataset and an associated model.

3.1.3.2. Labels

The first step into machine learning dataset building **[15]** is deciding the labels for which the data will be gathered. An investigation on the Internet on various websites that offer information about fruit **[17-20]** provided multiple lists of fruit names that are known across the globe. This method may not cover all the currently existing fruits but will cover the most relevant fruits because the labels come from popular fruit websites {**3.1.3.4 Page Rank**}. Having only known or well known labels in the dataset will affect what the application can classify in the end but user is highly unlikely to encounter a species of fruit that is not commonly found.

By utilizing a Python3 script, the multitude of lists of fruit found can be combined into one big label list. Since the initial lists can contain duplicate names the list needs to be cleaned. The removal of white spaces and conversion of all characters to lowercase makes the task of eliminating duplicates much easier by converting the list into a set that will remove any identical labels. Only trimming the white spaces is not a good solution because there are cases such as "*bread fruit* = *breadfruit*".

Reading the documentation on the same sources **[17-20]**, it can be easily seen that a single fruit can have multiple names, including: *regional names, popular names, scientific names*, etc. In order to completely eliminate duplicates from the original list the removal of semantic doubles is required.

Each name from the label list is be considered a duplicate if there is another name in the list that comes from the same fruit. Each fruit has a list of associated names for the process to work **(C.1)**. After the duplicate elimination process the label list was left with 350 labels out of the 650 of the original combined list.

```
["banana",
"red banana",
"green banana",
"yellow banana",
"musa sapientum",
"musa acuminata"]
```

C.1 - Banana Associated Names List

3.1.3.3. Data Source

Having the labels established, the next step is looking for a way of acquiring data. There are numerous fruit based datasets already built and available on the Internet {**2.3.1**}, but most of them contain small amount of labels or have a small quantity of information per fruit. Some approachable solutions in this case could be combining already existing datasets of fruit to obtain a bigger and better one, building a new dataset from scratch or maybe a combination of both methods.

The solution that starts by combining multiple datasets of fruit comes with the advantage of a lot less time invested in building but with certain consequences. Images coming from different sources tend to have different sizes and their target purpose might differ and also including that some of the better datasets are under different license terms that might make them unusable results into a hard to solve puzzle that might not even have all the pieces **[14]**.

Constructing and engineering a fruit dataset from the ground, on the other side, provides a lot of freedom in terms of data sources while also being able to aim for a specific target from the start. The obvious downside is the time consumed by tinkering, building and running the processes that will lead to the desirable dataset.

Considering the problems that are implied by fusing multiple datasets and the fact that the end result might not even be usable in the current scenario, the decision of building a dataset

from zero seems like the most efficient approach that balances quality and quantity with respect to the time coefficient, or at least, better than the alternative.

3.1.3.4. Search Engines

Building the dataset from scratch requires sources for data to be extracted from. Usually a group of professional photographers is necessary in order to build a good image dataset {**2.3.1**}, but they come at a big monetary and time cost. Since those resources are limited in the current circumstances the decision to import images from public domain is much more practical.

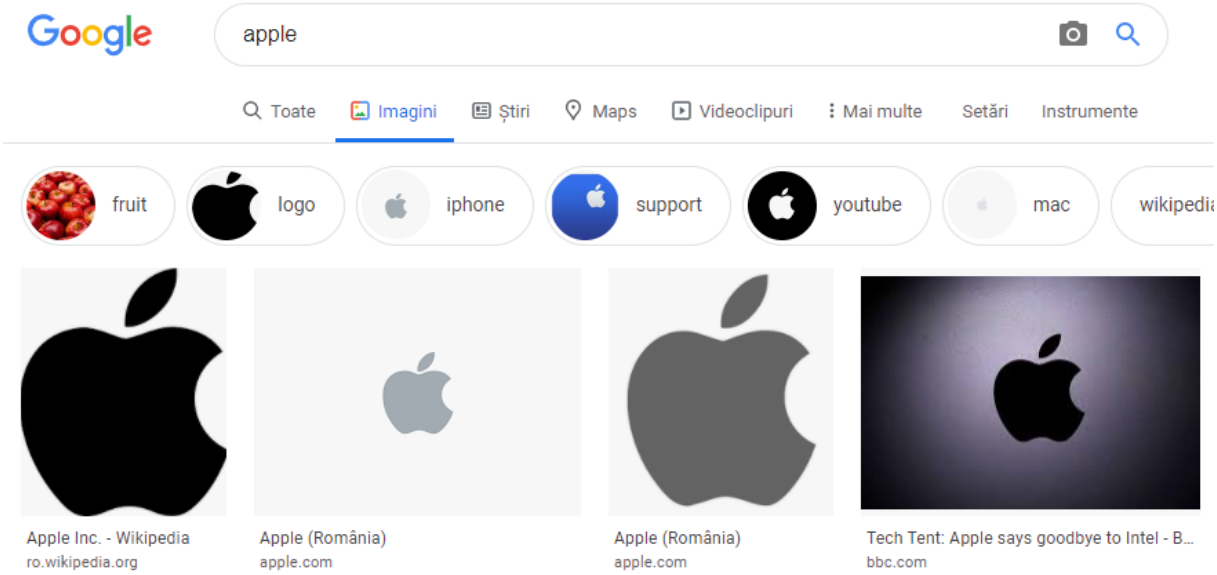
Search engines are generally a great source of public domain data, images or any other form, while also offering a very easy way to access it through a public API that features searching resources by entering keywords and strings that are analyzed through various algorithms.

A good example of a search algorithm is Google's PageRank[1] which works by counting the number and quality of links to a page to determine the importance of a said website. PageRank assigns weights of importance to each link contained within the whole set by recursively going through sources that refer to the page, the PageRank of the said sources being also taken into consideration **[21]**.

There are several problems that come with search engines as a means of acquiring data. The search results may vary and introduce a lot of noise in the data **(C.2)**. Finding a resource is sometimes harder than typing its name as the search string **(C.3)**.

C.2 - Blueberry Google Search Noise

[1] https://searchengineland.com/what-is-google-pagerank-a-guide-for-searchers-webmasters-11068



C.3 - ”apple” Google Search

Resources using search engines also inherit the problems that come from being data available through the Internet, one major downside being the volatility which translates to *broken resource links, corrupted files, different data formats* etc.

One of the upsides of the fact that resources are available through the Internet is that the data is presented in a HTML5 format, meaning that it can be parsed with a web crawler in order to find direct resource links from where the image data can be downloaded.

The chosen solution for gathering data to build the dataset on is to make a web crawler for different search engines that allow searching for images. Each fruit having a variety of search strings as input for the search engine.

In order to obtain the search strings that will be used for importing images, the list of names for each fruit {**3.1.3.2**} will be combined with various additional information that will provide more accurate searches. The additional information is simply a list of words that need to be attached to the already existing names utilizing a python script **(C.4)**.

```
["banana", "banana fruit", "banana tree", "banana fruit tree",
"musa sapientum", "musa sapientum fruit", "musa sapientum tree",
"red banana", "red banana fruit", "red banana tree"
"green banana fruit", "green banana fruit tree", "green banana tree",
"ripe banana", "ripe banana fruit", "ripe banana tree",
"musa acuminata ripe", "musa acuminata fruit", "musa acuminata ripe tree"]
```

C.4 - Banana Search Strings

3.1.3.5. Public Domain Data

Before building the dataset on public domain data, one should know that public domain data does not come without any restriction in terms of usage. Most data found on the public domain is under different types of licenses, which may or may not make them suitable or usable for certain tasks **[22]**.

Generally, using public domain data within machine learning algorithms, such as Neural Networks, resides in a gray area that varies from country to country **[22]**.

Usually data comes into public domain after its copyright protection lapses which makes most of it safe to use in Machine Learning projects. Many authors also publish their work under the Creative Commons (CC) license[2] which helps other creators to use their work under some specific rules set by them **[22]**.

Another valid argument that can be made is that data used for training a Machine Learning model cannot usually be recovered from the model alone. This makes it impossible for someone to tell if somebody trained their model on certain data or not.

The "Fair Use"[3] doctrine also allows copyrighted work to be used for certain applications (research, teaching, scholarships) without asking for the owner's permission. The doctrine also states that there are certain exceptions that come from the purpose, nature, amount, substantiality and effect of the application **[23]**.

This concludes that the dataset which is about to be built on public domain data must be under CC0 "No Rights Reserved"[4] **[24]**, meaning that anyone can freely obtain, modify and use it while also respecting each individuals images copyright terms set by the author.

Since the application will use a model trained on the dataset, the application should probably not be used for commercial purposes especially in countries that reside in the gray area in term of the copyright laws.

3.1.3.6. Importing Images

In order to obtain the images from the internet the usage of a web crawling algorithm is mandatory. To avoid the time investment necessary for building one from scratch the use of one of the already existing Python modules is advisable.

Selenium[5] is a Python3 module that is used to automate the interaction with web

[2]https://creativecommons.org/licenses/

[3]https://www.copyright.gov/fair-use/more-info.html

[4]https://creativecommons.org/share-your-work/public-domain/cc0/

[5]https://www.selenium.dev/documentation/en/

browsers **[26]**. At it's core selenium is built to work with webdrivers which are very similar clients to the common web browsers but they can be used within AI's to browse the web more efficiently. This module's API is the base of the web searching and data fetching unfolding in the importing algorithm.

The volatile data and links problem presented before {**3.1.3.4**} can be treated at this stage by creating an efficient algorithm for searching images. Errors that stop the algorithm must all be treated, but fortunately with specific "try and except" statements the script can run without crashing.

After analyzing a popular page that talks about the best image search engines **[25]** and taking the quality and quantity variables into consideration the image search engines that seemed the most appealing for fruit images were: *Google, Yahoo, Bing, Shutterstock, Unsplash* and *Getty*. The majority of images found on them also tend to have the same format "jpeg".

Each search engine requires an unique selenium algorithm **(Algorithm 1)** because the way the image data is presented in HTML differs and also because of how the interaction with the said search engine works **(C.5)**.

C.5 - Image Src Location in Google Searches

In order to speed up the process of gathering data each algorithm will be run on a different thread with certain shared variables that allow the images to be stored simultaneously.

Already considering the amount of noise to be high in the resulted dataset, the amount of 10^4 images per label was considered by purely speculating on the bad scenarios and conclud-

ing that only 10% of the resulting dataset will be usable, the goal being at least 10^3 good image data.

The condition for stopping all the import algorithms running on multiple threads for each fruit label is the passing of one hour of search. The hour of search was calculated by testing the import script on some labels initially to see the average amount of time needed to import 10^4 images per label.

Algorithm 1 Import Google Image Data

```
1: Open chrome webdriver
2: Access the google image search page while avoiding the user agreement
3: Access the google search bar by HTML5 element where name = ”q”
4: Type the current fruit label string and press ENTER and wait 2 sec
5: While not enough images are found and the current search still has images
6:     items ← all elements with id = ”islmp”
7:     image batch ← all sub elements from items that have the tag ”img”
8:     src batch ← all values of ”src” attributes from image batch
9:     valid srcs ← all valid and accessible links from src batch
10:    final srcs ← all download links from valid srcs that are not in multithread safe set
11:    for each element from final srcs
12:        try to download the image from element under global counter.jpg name
13:        if any download error appears
14:            skip the current element
15:        else
16:            increase the local image counter
17:            increase the global image counter
18:            add the element to the multithread safe set
19:    if no image data was fetched from last batch
20:        try to press the button ”load more images” that has class name = ”mye4qd”
21:        ignore errors that are thrown
22:    scroll down in the webdriver window
```

3.1.3.7. Automatic Image Filtering

Knowing what the usage of the data will be, filters can be built in order to remove the unnecessary images and noise. Some filters can be applied at the runtime of our data importing algorithm while the others require the image data to be able to work.

Duplicates

Duplicate images decrease the efficiency of training the model **[10]** and must be removed from the dataset.

Queries done on the search engines can contain images that have the same **src**, which means that those images will without doubt result in a duplicate when downloaded. To filter those types of duplicates out a multithreaded set can be used within the algorithm **(Algorithm 1 - Line 10)** to check if the resource found at a specific **src** has already been downloaded.

Sometimes the same image can be found at multiple **src** and this type of duplicate cannot be treated at import time. The simple way of solving this problem is comparing all the unique unordered same sized image pairs in the dataset and purging the doubles. This method turned out to not be very efficient because the complexity for this is:

$$\mathcal{O}(images\_number \times images\_number \times (image\_width \times image\_height + IO\_load\_time))$$

The values of the product $image\_width \times image\_height$ in the current dataset is averaged at 257×304 {**3.1.3.8**} which means that in the worst case scenario when comparing two average sized images there will be a total of 78×10^3 operations. The resulted dataset contains 2.6×10^5 {**3.1.3.8**} images, which means that the whole procedure will have around 2×10^{10} operations. Adding on top the IO operations, which are done on a hard disk {**5. Hardware**}, that are used for loading each individual image, the resulting algorithm is not feasible considering the time available.

There are no ways to skip comparing all pairs of images because the desired result of eliminating all the duplicates will be sacrificed in the process. To improve the time needed for the algorithm a compromise with the allocated memory must be made: each image will be condensed with the help of a hash function into a much smaller amount of information that will be in the memory.

The total number of images 2.6×10^5, which is considered pretty small for generating hashes, allows for the use of a faster hash function but weaker in terms of collision. In this case

MD5 from the Python3 Crypto [6] module will be used. The collision chance for MD5[7] is around 1.47×10^{-29} and considering the size of the dataset the chances for this method obtaining two identical hashes for two different images is almost impossible. Going forward with the calculation gives the chance 6.76×10^{10} out of 1.47×10^{-29}, the end result being 4.6×10^{-19} chances of obtaining two identical hashes from two different images.

Concluding the optimization, the 78×10^3 average computations needed to compare two same sized images transformed into comparing two 128bit variables. The new comparison method helped in gaining the speed to filter out the duplicates in 10 minutes worth of time, this considering the available hardware {**5. Hardware**}.

Image Format

One of the criteria on which the current search engines were chosen was image representation format, in this case most of the images being under the "jpeg"[8] format.

The process of eliminating the images, which are not under the standard "jpeg" format, can be done at import time by examining the direct link provided by the "src"[9] attribute because the specific image search engines, chosen for importing in this case, include the image format within it.

Corrupted Data

Sometimes when downloading data parts can become corrupted, thus making the entirety of the item downloaded unusable. The corruption can happen due to various means, hardware or software, ranging from discontinuity in internet connection to too many corrupted bites for the recovery to be fully successful.

To eliminate such cases of corrupted data, the importing must be already finished. The process of purging is as simple as utilizing a Python3 module that focuses on processing image data, such as Pillow, which will be used to open each image. If the image is corrupted there will be an error when trying to open it as an image. By parsing through every image and catching the errors that appear, any corrupted file can be deleted. From the 2.6×10^5 {**3.1.3.8**} images approximately 300 (0.001%) were removed by this filter.

[6] https://pypi.org/project/pycrypto/

[7] https://www.ietf.org/rfc/rfc1321.txt

[8] https://en.wikipedia.org/wiki/JPEG

[9] https://html.com/attributes/img-src/#src_and_srcset

Image Dimensions

The reason of building the dataset is for training different Convolutional Neural Network models on it, which generally use the same shape (dimensions) for the input data in a session of training. Knowing this and the fact that the data imported from image search engines does not respect the same shape, it can be foretold that the data will need resizing for it to be fed to the model.

The problem of resizing images that are very small is the noise which the process introduces in data, the bigger the ratio between the image size and the resizing dimensions the lower image quality.

Analyzing the results of importing, it can be observed that the image width and height tend to be close to each other numerically, most of the images being close to squares. Following this empiric result, it can be concluded that images that do not respect a certain ratio between the width and height can be safely eliminated since the result of their resizing will introduce a lot of noise, taking into consideration that the target for resizing will be the average of dimensions. In the current scenario all the images that had their bigger dimension more than twice as big as their lower dimension were purged **(C.6)**.

C.6 - Too Wide Acerola Image

Knowing beforehand that multiple experiments on image sizes ranging from 16×16 to 256×256 will be conducted and adding on top the facts about resizing from small to large images, it can be safely assumed that image sizes under certain thresholds must be eliminated. The thresholds will be determined through empirical testing in resizing small size images to 256×256 images **(C.7)**.

Visually analyzing the resized images and following the paradigm ”image type/origin can be discerned by the human eye”; the conclusion for the imported data was that all images that have the width or the height stricly under 64 pixels must be removed from the dataset hence the data does not follow the said heuristic.

C.7 - Resizing Acerola Image From X to 256

3.1.3.8. Resulted Dataset & Statistics I

Labels & Quantity

There are 2,610,914 images in total in the dataset amounting to around 67 gigabytes of data. From the initial 350 labels only 293 remained within the dataset because some of the fruits were too peculiar and the chosen image search engines had little to no data about them. From the 293 labels that were found: 204 labels had under 10^4 images per label, the previously set target {**3.1.3.6**}, and 89 above.

Time

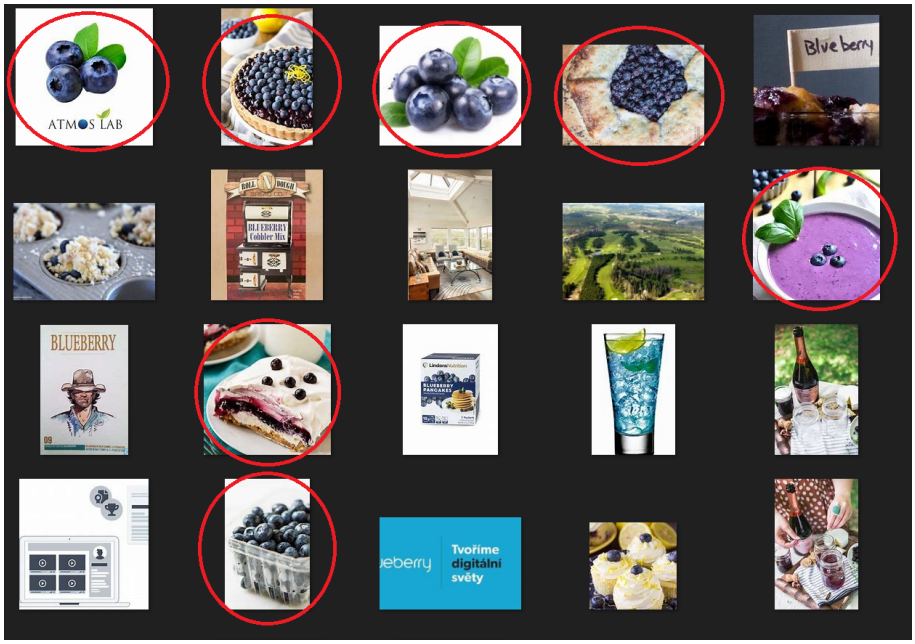
The time needed for writing a first viable and functional iteration of the import script took one week and the import of all the data with the import time filters applied took approximately 30 days. The filters that are applied after the import phase were constructed during the importing of the data and their application took only one day. The total time needed for this iteration of the dataset amounts to one month and approximately one week.

Other important remarks about the time are that considering the hardware resources available {**5. Hardware**} and the limited human resources only around 8-12 fruit labels were completed daily; the number varying depending on the uncontrollable variables such as the unstable or slow internet connection which sometimes required manual intervention to restart the import script from a certain label.

Even the last iteration of the importing script had no way to remember the exact location and image on the search engine if the internet connection happened to drop; this implying that outside intervention was needed each time the unfortunate event happened.

Quality

Regarding the quality of the images; as can be seen **(C.8)**, the data contains a lot of items that seem to be related to the fruit or related to the meaning of the search string but do not contain the fruit in any shape or form, this representing a large volume of noise in the dataset and reducing its overall quality.



C.8 - Blueberry Label - Images Containing Blueberries

Statistics

Table A	Average	Median	Standard Deviation
Number of Images per label	**8,942**	**7,517**	**5,736**
Image Width	**258**	**249**	**39**
Image Height	**304**	**304**	**38**
Missing Images (target 10^4)	**5,953**	**6,010**	**2,248**

The average number of images per label tends to respect the 10^4 target {**3.1.3.6**} but taking the standard deviation into account the dataset is far from balanced. Considering the

bracket for missing images, which was applied only to the 204 labels under the number quota, it becomes clear that some labels might have only 10% or even less of the required amount, while others may be numbered at double or even triple the necessary quantity hence the average number of images and missing images are both high when comparing to the target. **(Table A)**

The average amount of pixel data found within an image meets the criteria set before {**3.1.3.7 Image Dimensions**} with somewhat low variance between items, which means that resizing will result in less noisy data. This result comes from the raw images found on the internet that follow a standard for dimensions and the automatic size filters that were applied onto the dataset. **(Table A)**

3.1.3.9. Manual Image Filtering

Problem

The resulted dataset was not in the desirable range of quality and most likely trying to apply Machine Learning algorithms on it will most likely result in very low accuracy; which was expected from the importing step {**3.1.3.6**} considering the nature and provenience of the data.

To obtain a better quality, the removal of any image without the right fruit content was mandatory. Since the dataset contained labels that had as low as one to two thousand images {**3.1.3.7**} and there were no labels without bad items, a new target for quantity needs to be established, numerically speaking around one thousand images per label.

Filtering the images that did not contain fruits out can be done in two different ways: automatically or manually.

Automatic Filtering

Automatic filtering implies the use of an AI trained to classify images as containing a fruit or not containing a fruit. The problem of this approach is that the AI needs to also classify the fruit, for example images of apples cannot reside under the cherry label. The fruit recognition, even without the classification, is approximately the goal this project is trying to reach in terms of the Machine Learning work. The usage of a fruit recognition tool in order to build a fruit recognition tool feels like cheating and as a result the automatic filtering removed itself from the possibilities list.

Manual Filtering

Manual filtering or filtering done by hand with the help human labor can be achieved in many ways nowadays. One commonly used method is a system like the ReCaptcha from *Google* **[27]** which consists in making people classify images in order to obtain access or prove they are not AI's. This way google classifies images while also offering a service to the clients that need help in distinguishing between human and AI controlled users that access their services.

The problem with this kind of approach is that the span of time necessary to fully classify and sanitize the dataset is highly dependable on how many people there are and how many times will they be interacting with the system. The solution to the problem is to make the fruit classification system popular or integrated in a popular service accessed frequently by people and then wait for all the images to have a pretty reliable fidelity score of whether they contain the specified fruit or not.

The other available solution is to find a group of people that will be able to help filter the dataset. The advantages of this method, comparing to the above one, are the time gained of not making the system popular and the increase in fidelity since an algorithm for fruit image selection can be applied. The team of people can also be volatile consisting of volunteers that want to help for the cause.

Testing and Conclusions

The manual filtering **(C.8)** of one fruit label with initially 2×10^4 images takes around 30 minutes, empirically tested on some higher quantity labels. The calculations being purely speculative it is needed to assume one of the worst case scenarios (one to two hours per label) in order to come with the time frame needed. Having 293 labels, this amounts to 36 to 73 days of work, taking into account only 8 hours a day.

The first method of manual filtering, while being the easiest way when it comes to work investment, does not guarantee the results nor the time frame needed to get to them. The second method will bring the project to a good final state and requires the building of an image selection algorithm, finding people and digging through the 293 labels containing the 2.6 million images. To complete the second method, in the worst case, it will take around 3 months if only one person participates.

At this stage the project only has 6 months left until the end. The conclusion is the latter, start sorting through the dataset utilizing a selection algorithm until no labels remain and while sorting also find people willing to help to reduce the time period needed. This method

will at least guarantee the success of building a better dataset within the limited amount of time that remains.

3.1.3.10. Image Selection Paradigm

The next step is to build what will be the process of iterating through all the labels whilst filtering their images while also being able to manage the work handled by additional party members that come and want to participate.

The task will be broken down into small tasks, each having the target of sorting one unique fruit label, amounting to 293 total tasks. Most of the fruit labels are not easy to encounter in a daily life of a human, so the decision to split the list of tasks into two lists: one with easy to sort and the other with hard to sort fruits. An easy to sort fruit might be the ”apple” or ”banana” while a harder task might be to sort ”white sapote” or ”desert fig”.

The filtering strategy for the images will be formed by differentiating a good image candidate from a bad one. Following this logic an algorithm of selection can be built by breaking the whole process into simple steps that allow one individual person to make accurate decisions on what will the fruit label allow or not. While starting with a simple and logical algorithm worked for a single person, for a team the expansion was absolutely necessary for each member to understand the selection concept and to make a good selection. The paradigm suffered a lot of changes and had multiple iterations but the end refined result follows among the steps of:

I. Gather Information

To be able to select viable images of a certain label certain information about the fruit and its life is required; one should be able to answer the following questions, in no particular order of importance, in order to start filtering the image data:

- *what is the habitat or habitats that the fruit might grow in*?
- *what are the colors and shapes the fruit might have in any stage of its life*?
- *is the fruit usually found in groups*?
- *how does the interior of the fruit or its slices look like in any stage of its life*?

To acquire this kind knowledge any method can be followed, the most used ones including searching information and images on the Internet or looking up botanic books.

II. Select Fitting Images

The second step is to start iterating in any order utilizing any type of visualizer for the images and selecting items that respect the next criteria, until the thousand target or the end of the label data is reached. If any of the items falls in a gray area, it should be saved separately in case there is a need for more data at the end of sorting. The most popular applications used for iterating and visualizing being in this case Windows Explorer and respectively Paint.

II.I. Criteria of MUST

All images that are to be selected have the requirement to follow the following properties:

- have at least 50% of the pixels occupied by the respective fruit's data;
- (OR) contain at least the entirety of at least one fruit from the label;
- (OR) contain at least the entirety of at least one slice of the fruit from the label.

II.II. Criteria of MAY

All images that are to be selected can have one or mutiple of the following properties:

- contain the fruit in any stage of its life;
- contain any type/color/form of the label;
- have the background representing the natural habitat of the fruit counted towards the 50% fruit information;
- contain other items other than the fruit and it's related habitat;
- be formed through the combination of multiple smaller images, if the data is scarce for the current label; these falling usually under the gray area category.

II.III. Criteria of SHOULD NOT

All images that are to be selected should not have any of the following properties:

- contain any other fruits or fruit parts than the ones coming from the label;
- contain the fruit represented by a drawing and, not a natural instance of the fruit;
- have obvious filters applied to the image that alter the fruit information.

III. Corner Cases

There may be situations where an individual cannot be sure what to do with some items while also not having enough data to hit the 1,000 mark set on the label. For these peculiar

situations all the team participated and voted on what to do with them.

By the end the team was composed of 6 members, each having contributions ranging from small to large to both the selection paradigm and the resulted dataset. **(Team Table)**

Team Table					
Team Member	**Gender**	**Age**	**Occupation**	**Number of Labels**	**Status**
Minut Mihai	**Male**	**22**	**Student**	**164**	**Filtered**
Minut Emilia	**Female**	**16**	**Highschool Student**	**78**	**Filtered**
Dumitriu Alexandru	**Male**	**21**	**Student**	**12**	**Filtered**
Profir Petrisor	**Male**	**21**	**Student**	**6**	**Filtered**
Loghin Vlad	**Male**	**21**	**Student**	**1**	**Filtered**
Onu Robert	**Male**	**21**	**Student**	**1**	**Filtered**
All Members	**-**	**-**	**-**	**31**	**Removed**

3.1.3.11. Resulted Dataset & Statistics II

Labels

During the manual filtering phase many of the labels proved to not contain any image containing useful fruit data and needed to be removed entirely; a good example being ”african apply cherry” that had data regarding the labels ”apple” and ”cherry” but none fitting under the whole name. As a result some of the labels were removed entirely from the dataset.

A mistake in a previous stage {**3.1.3.2**} was to not check if the same fruit label can be found under different names depending on the region. For example the fruit ”chinese bayberry” is an instance of the ”bayberry” fruit that can be found mainly in China and while the visual difference is almost nonexistent **(C.9)** between the two it makes sense to merge them under the same generic label, namely ”bayberry”.

After filtering and going through all the removals and merges the final dataset came to have a total of 262 distinct labels from the 293 resulted from the original importing.

Quantity

The resulted dataset now amounts to 225,640 images which translates to approximately 2.7 GB of data. The dimension of the dataset decreased by a magnitude of 9, which is very

C.9 - Bayberry vs Chinese Bayberry

realistic considering the target was to reduce the 10,000 images to 1,000 images per label and also the removals.

From the 204 out of 293 labels that were under the target previously, only 100 out of 262 have now under the 1,000 milestone, which translates to an increase in fruits that have enough data from 30% to 61%.

Time

The time cost for the whole manual filtering process was roughly 7 to 8 weeks. Starting with two to four fruits per day and slowly adapting to the algorithm and the algorithm itself, the team increased its efficiency gradually and by the end a day would amount to at least 12 fruit labels sorted out.

Quality

Even though the amount of information has been lowered considerably, the increase in quality of the data made the process of manual filtering very efficient **(C.10)**. All the fruit images, considering the norm implied by the filtering paradigm, contain now the correct fruits, habitats, trees, bushes and other forms of data that helps greatly in distinguishing the many existing labels.

Statistics

Comparing the statistics made on the new dataset **(Table B)** and the ones made before manual filtering **(Table A)** while also considering the decrease in the order of magnitude it can be clearly stated that the average number of images per label remained the same, but the median went up by about 33% and the Standard Deviation halved. This result proves that the balance of the dataset has risen along with the number of labels that respect the target number of images.

C.10 - Blueberry Images after Manual Filtering

Table B	Average	Median	Standard Deviation
Images	861	1007	276
Image Width	213	209	19
Image Height	262	255	30
Missing Images	580	567	258

Analyzing the new average amount of pixel data of an image the information loss coming from the filtering can be estimated at about 14%, but considering the increase in quality per pixel makes it is justifiable.

While the labels that fail to reach the target amount has decreased, the ones with problems do not seem to be improved after the filtering, their numbers remaining almost the same. This, with the addition of the still high variance in image numbers per label, proves that the paradigm chosen for filtering did not take into account the balance of the dataset. While it fixed the problem of shortage of data by reducing the target, it did not fix the problem for the labels that did not have many items or had too many invalid ones,

3.1.3.12. Problems

Amount of Data

The current dataset, even after multiple iterations, while is viable for machine learning is still not in a perfect state. The amount of images per label, considering the number of labels is still very low, implying that the machine learning algorithms will be more inefficient when

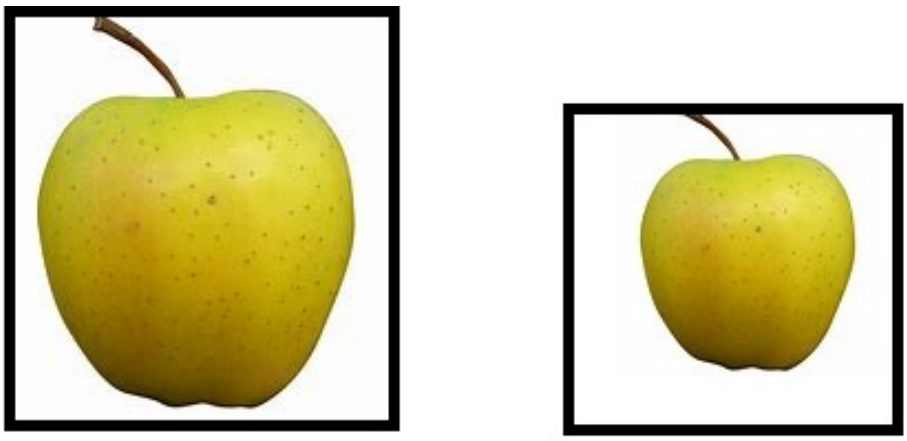
trying to separate the fruits. Most likely augmenting the dataset will be required in order to raise the model quality.

Balance

The label imbalance problem, while in a better state, is still present and will most likely damage the accuracy of the model on fruits with the lowest and highest amount of information, some being overfitted while others being in a deficit of data.

Similarity

During the process of manual filtering many team members have reported that they had found images of the same fruit with different image dimensions or of the same fruit but in different angles **(C.11)**. While this is not currently a big problem it can become one after the resizing of the dataset because there may be cases where duplicates might appear. Even if there will be no duplicates after resizing there will still be bigger chances of overfitting the model since the data contains similarities.



C.11 - Same Apple - Similar but Different Images

3.1.3.13. Conclusions & Future Work

Importing - Search Engines

While importing data from search engines has the advantages of building a dataset free of monetary charge and relatively fast when compared to producing each image it has some major downsides:

- the dataset cannot be used, according to the legal terms, as a resource for profit in commercial purposes and may not be used in all kinds of scientific projects, depending on the local laws of copyright;

- through the raw data coming from such import sources, a lot of noise and meaningless items can be found which will absolutely require filtration and balancing;

- some of the labels might have little to no items available in the public domain resulting to their removal from the dataset.

To further improve the dataset, already existing datasets might be integrated if their license allows to do so. This operation could improve the balance of the dataset by filling up spaces that labels have and could also increase the total quantity or average quality of the dataset.

Filtering

The automatic filtering has, in the current scenario, improved the quality of the dataset by eliminating big chunks of data considered to be noise. Other options of automated filtering could be added to further develop the quality of the dataset; such as AI based filters that already know which elements are not suited for machine learning training algorithms or that know to recognize images containing fruit data from the rest.

The problem with similarity could be solved by building a filter that compares two different sized images and deletes duplicates based on the similarity percentage.

The manual filtering, while slower than a fruit identification AI and while requiring a lot more human labor, also provides a big boost to the overall dataset quality and with the correct filtering paradigm and team could even be better than the alternative.

Remaining Problems

The problems that are still present {**3.1.3.13**} could be fixed by further iteration on the dataset. While the amount of data and overall balance can be fixed by augmenting and integrating other datasets {**3.1.3.3**}, the similarity of two images can be difficult problem to solve considering the anti cheating policy that the project follows.

There may be other problems present but unknown in dataset that might be revealed in further stages of development of the application that are highly dependent on its quality.

Final Dataset

Initial researches and investigations showed that the structure and hierarchy of the plants changed the concept for the application to have multiple more accurate models. The sources of the fruit data were image search engines, containing public domain data, reachable through the Internet. The initial raw data has been put through multiple layers of filtering, each with their own preconditions and results, in order to get to the last iteration of the dataset built thus far.

From the initial task of building a perfect dataset to classify all plants the decisions and compromises made lead to the result of having a dataset of a total of 262 labels containing only fruits from the plant family. Comparing to other datasets available on the internet, the 262 labels containing around 22×10^4 total images is a pretty good result in terms of label number and average in terms of amount of data. The imbalance, similarity and the limited amount of data problems that are still lingering in the dataset should help get the answer to how to further improvise the dataset.

Fruits 360 Comparison

Comparing the end result with the ”Fruits 360” dataset **[29]** it can be stated that the obtained dataset is very unbalanced.

The view on what defines fruit data or information is much sparser due to the paradigm of manual filtration {**3.1.3.10**} which can classify fruit data as whole fruit, fruit slices, natural habitat, multiple instances of the same fruit etc.

While fruit species coverage is much broader in the dataset built here the subspecies ramification is close to none since all types and subspecies of a clade of fruit are treated under the label of the original fruit. For example all 13 types of apples presented by the ”Fruits 360” dataset are under the same label, namely ”apple”, in this dataset.

The similarity of images, while still present {**3.1.3.12**}, is on a much smaller scale which could in theory help in classifying fruits in real circumstances much easier.

Dataset & Scripts Location

The dataset and useful information regarding the labels and other variables related can be found at Kaggle - Fruits-262 **[28]**.

The python scripts used to organize the data are in the same location under the scripts section. All of the scripts utilized throughout the project reside on a GitHub repository[10].

[10]https://github.com/Elaech/Fruits-262

3.2. Fruit Image Classifier

3.2.1. Convolutional Neural Networks

Pros

As it can be seen, from other projects {**2.3**} and from the theory {**2.2**}, Convolutional Neural Networks not only excel at breaking down the image classification problems into smaller ones but are widely used to do so.

Considering the fact that most of the results of the people mentioned {**2.3.2**} are about CNNs, the easiest take on building an image classification pipeline utilizing specialized tools and techniques is having them at the core.

In the majority of cases, the training and testing pipelines for a said CNN model is mostly the same. This will reduce the time needed for building the two separate procedures.

Having already made comparisons between the "Fruits-262" {**2.1**} and "Fruits-360" **[29]** datasets, it would be much easier to make statistics and draw conclusions since the "Fruits-360" paper **[31]** examines the potential of CNNs on a fruit dataset.

Cons

Other types of machine learning approaches might be as efficient if not better than Convolutional Neural Networks for building a good model.

Usually training bigger CNN models ca be demanding in terms of hardware resources and can also take a long time to train. Experimenting with them might not leave enough time for fully exploring the potential of other types of machine learning algorithms.

CNNs are one of the machine learning methods that require a lot of data manipulation to mold the dataset information into trainable and testable material.

Conclusion

Considering the remaining time for the whole project being around 3 months, there are two choices: vaguely explore multiple ML algorithms or focus on one family of algorithms in order to obtain a better model in the end.

The decisive factors were the available information and statistics provided by "Fruits-360" dataset contributors and also the knowledge accumulated. The final decision was to focus solely on CNN based models and try to obtain a good model that will generate results that could be compared with the other fruit dataset statistics.

3.2.2. Organizing Data

The fruit data is currently stored in different directories with the label being the directory basename. Images are stored as individual units on the hard disk {**4**} with different dimensions in the ”jpeg” format.

3.2.2.1. Data Splitting

Splitting Intervals

In order to train a CNN model, data for training and validation has to be provided. After finishing the train process the model can then be tested against data that it has never seen before, namely test data, in order to produce relevant statistics and results.

Having the entire data in one big piece means that there needs to be a decision regarding how much information to attribute to each sector: training, validation and testing. Observing other experiments {**2.3**}, the majority of the data seems to be attributed for training while the minority is split between the validation and testing with different magnitudes, sometimes even zero information, eliminating a step altogether.

The final decision was to go with 70% of data from each fruit for training (rounded up), 20% for validation (rounded up) and the remaining will be used at the end for testing the data.

Splitting Process

In order to speed this process, instead of working with image data which requires loading time, the image paths will be used. For splitting to work effectively the path needed to be unique for each individual image while also providing information regarding its label.

Windows 10 does not support duplicate file naming within the same directory and also the ”Fruits-262” dataset holds all of the images in directories named after the their respective label. An example of such path would be *”../apple/303.jpg”* which contains the label and the unique image path.

Utilizing a python script the data for each fruit was split with respect to the established percentages and added to one of the respective global lists of training, validation or testing.

3.2.2.2. Resizing Data

Dimensions

CNN models accept as input only a set size **[12]** and as a consequence: all the data that will be used for training, validation or testing needs to follow a standard for the size.

The current dataset contains images that have a high variety of dimensions. Resizing them to the *average_width x average_height*, meaning 213 *x* 262 {**3.1.3.11**}, would give result in the least information loss per image.

The above theory, while being correct, it does not take into consideration how CNN models work and why it would be an advantage to have multiple sets of dimensions for resizing the dataset. Given an image of size *w x h*, the image will be resized throughout the model multiple times because of the pooling layers, which usually are set to have a kerner of 2 *x* 2. The problem comes from the fact that the images can only have positive integers as dimensions, which implies losing information between the pooling layers in cases which integer division is not an option.

The end result needs to be a compromise between the information loss and the scaling of dimensions. Following the generality of 2 *x* 2 pooling layers it can be concluded that the size of the images will be gradually divided by 2 within the model.

Following the minimum information per image admissible {**3.1.3.7**}, the smallest image dimension should be around 16 *x* 16.

Adding the logic behind the image processing during the model, the resizing dimensions should be as close as possible to 16 *x* 16, 32 *x* 32, 64 *x* 64, 128 *x* 128 and 256 *x* 256.

To calculate the perfect resizing values for the lowest information loss the minimum of the following function needs to be calculated:

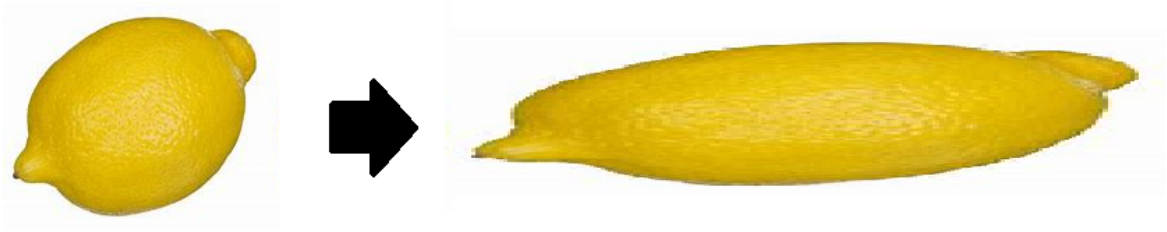
$$f(dimension) = |\ average\_information\ -\ dimension\ x\ 16|,\ dimension \in \mathbf{N^*}$$

The above function represents the information loss according to the smallest resizing dimension but with respect for the highest dimension. For example: resizing an image to width 16 will come with the information loss of $|213 - 256| = 43$ at the highest dimension (also being the highest information loss dimension).

Solving the equations regarding the minimum for each the width and the height gives the result that the images should follow the dimension standard: 13 *x* 16, 26 *x* 32, 52 *x* 64, 104 *x* 128 and 208 *x* 256.

Image Ratio

Considering the subject of the dataset being fruits, the conclusion is that altering fruit image width and height ratios can completely destroy the semantic value for certain fruits **(D.1)**.



D.1 - Lemon Image Resized - Modified Ratios

A simple solution to this problem is adding blank padding to the final images to be able to maintain the original image ratio while also respecting the dimension standard.

Resizing Process

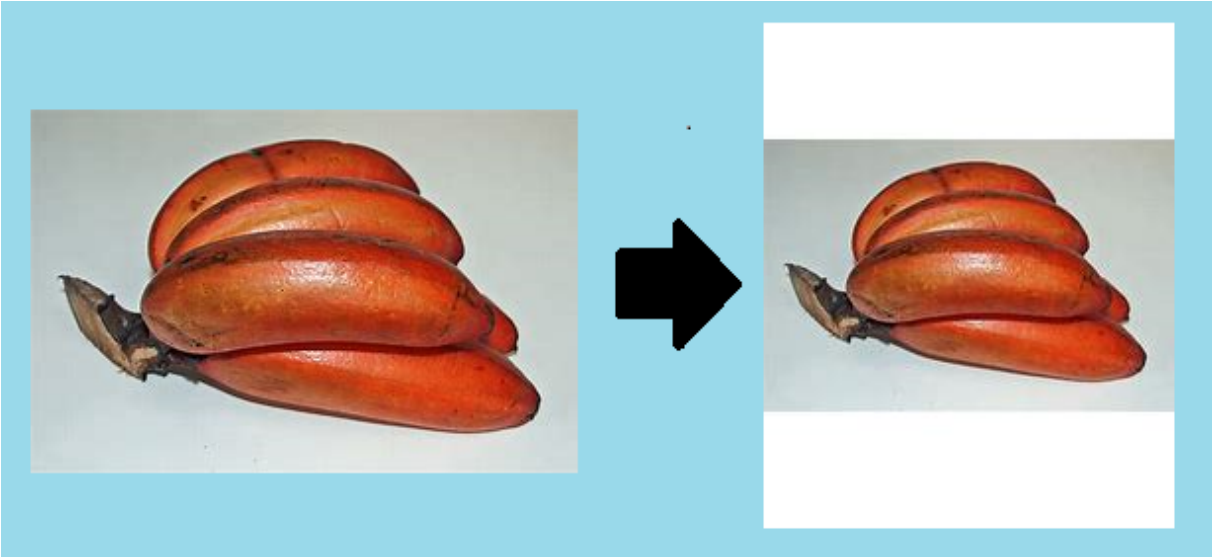
The problem of resizing images, now resides in questioning whether an image should be increased or decreased in size. There is only one choice and must be done only once for each individual image, this being a consequence of the keeping of the ratios rule.

The method used will follow the principle of increasing the size of images that have both the width and height lower than the target dimension while decreasing the size of images that have at least one dimension, width or height, bigger than the goal **(D.2)**.

To form the final resized image from an input one the following steps will be made:

- create a blank image that has the target dimensions
- resize the current image conveniently
- insert the resized image at the center of the blank one
- save the result to the designated directory

At the end of the process the dataset will have been resized with respect to the resizing standard. Each set of dimensions will have its own directory mimicking the structure of the initial dataset with images found under directories named after their labels. These resulting variants of the dataset can also be found at Kaggle - Fruits-262 **[28]**.



D.2 - Banana Image Resized

3.2.2.3. Efficient Data Storage

Loading Times

Before proceeding with training a model, the data must be loaded and stored in structures accepted by the CNN as input. This process implies the reading of data from the storage device {**4**} and the molding it into specific data classes.

Having multiple sets of dimensions for the dataset, the loading time has been tested multiple times for each sets of dimensions **(Table C)**. The main cause for the loading problem is the storage device, in this case an HDD {**4**}. Hard disks are usually slow at reading data from different location within the memory but relatively fast at reading continuos data from the memory.

At this moment the images are stored individually which is likely to make the loading process slow. The solution is finding a way to enforce the continuous storing on the hard disk.

Loading Capacity

While doing the loading tests, another problem has been observed: loading the image data in the format needed for training occupies around 12GB on the RAM. The 75% usage is not feasible because complex CNN models also occupy a decent amount of memory to store the weights.

Solving this problem would mean dividing the data in smaller packages that will be loaded or preloaded during the process at fixed time intervals.

Table C								
Dimensions	**Load All Data**	**Efficient Load All Data**	**Efficient Load One Pack**	**Train Packs**	**Valid. Packs**	**Test Packs**	**Size HDD**	**Size RAM**
13x16	10s	1s	1s	1	1	1	81MB	400MB
26x32	31s	4.7s	4.7s	1	1	1	300MB	1GB
52x64	1.5min	15s	15s	1	1	1	1GB	3.5GB
104x128	7min	2min	27s	3	1	1	1.1GB	4GB
208x256	30min	10min	36s	12	4	2	1.1GB	4GB

Efficient Storage Solution

For data to be stored continuously within the memory there needs to be at least a space big enough to fit the whole packages. To ensure this, a defragmentation process of the hard disk will be run.

The format of storing the data will change from individual ”jpeg” to mass package storing under the ”npz”[11], a popular Python3 Numpy[12] format. This will allow multiple images to be put into a big array of data, the shape type being $(width, height, channels, array\_size)$, and then binary serialized on the hard disk into one or multiple packs of data, depending on the dimensions and size.

Having the new format for the packs of data **(Table C)** has improved the loading times and allows for less strain on the RAM memory in the training process, which will mean the possibility of training better models.

3.2.3. Model Training and Feasibility

Having the data prepared, the next step will be defining a simple model architecture scalable in terms of input dimensions. The model will be trained on the entirety of the dataset in order to determine the feasibility in terms of time and learning, the goal being to determine the possibility of using more advanced models.

Simple Model				
Units	**Layer**	**Kernel Size**	**Activation**	**Layer Active Condition**
8 filters	Convolutional2D	3x3	Relu	padding, 208x256x3
-	MaxPooling2D	2x2	-	padding, 208x256x3
16 filters	Convolutional2D	3x3	Relu	padding, >= 104x128x3
-	MaxPooling2D	2x2	-	padding, >= 104x128x3
32 filters	Convolutional2D	3x3	Relu	padding, >= 52x64x3
-	MaxPooling2D	2x2	-	padding, >= 52x64x3
64 filters	Convolutional2D	3x3	Relu	padding, >= 26x32x3
-	MaxPooling2D	2x2	-	padding, >= 26x32x3
128 filters	Convolutional2D	3x3	Relu	-
-	MaxPooling2D	2x2	-	-
256 filters	Convolutional2D	3x3	Relu	-
-	Flatten	-	-	-
128	Dense	-	Relu	-
262	Dense	-	Softmax	-

3.2.3.1. Simple Model Structure

The model structure **(Simple Model)** was built by following a classic structure for CNNs. The beginning is formed by stacking multiple pairs of convolutional and pooling layers, which will act as filters for the image features. The number of layers is adjusted so that the end layer will produce the 4 *x* 2 *x* 3 output shape, this implying that the network will have a low amount of parameters to train. The number of filters grows with the depth of the convolution, thus allowing for more efficient training **[12]**.

The final layer, the output, will provide the final results of classification. Each of the 262 output units represents a fruit label and contains the probability of the said label representing the input image. This is due to the activation function, namely Softmax[13], which transforms the final weights into percentages according to their size. The rest of the layers utilize a standard Relu[14] activation function.

[11] https://numpy.org/doc/stable/reference/generated/numpy.savez.html

[12] https://numpy.org/

[13] https://en.wikipedia.org/wiki/Softmax_function

[14] https://en.wikipedia.org/wiki/Rectifier_(neural_networks)

The number of filters and units throughout the model also were adjusted for a better training speed, but a weaker power of classification. The reason for this is that this model's goal is testing how it handles the substantial amount of data that the fruit dataset has.

3.2.3.2. Training Tests

In the following tests the model will be trained using the Adam[15] algorithm as the optimizer and the Categorical Crossentropy[16] function for computing the loss. These choices were natural for our conditions: Categorical Crossentropy is good at separating the value of each label compared to the other ones while the Adam algorithm is generally used and performs well on the majority of scenarios {**2.3**}.

Other choices for the loss function can be made; for example the Triplet Loss[17] function is very good at calculating loss on instances with many different labels. Since the dataset has 262 labels, this loss function might be better than the standard Categorical Crossentropy but requires additional testing time and model structure managing to be effective.

The table **(SMTT)** contains statistics regarding the training of the model. The tests, while being in the same table, have different goals and meanings which will be covered in the following sections. Fields marked with "∼" had no meaning due to the nature of the tests.

Simple Model - Training Tests (SMTT)					
Index	**Input Dimension**	**Epoch Time**	**Epochs Number**	**Train Accuracy**	**Validation Accuracy**
1	13x16	12s	1000	40%	22%
2	26x32	28s	1000	60%	29.5%
3	26x32	28s	3000	99.4%	∼
4	52x64	1min	1000	65%	33.33%
5	104x128	4min	250	∼	∼
6	208x256	20min	50	∼	∼

Time

The first time goal was to measure the time it takes for a model on average to complete an epoch on each dimension set. By feeding the data to the model up to 1000 times at a moderate

[15] https://arxiv.org/abs/1412.6980

[16] https://en.wikipedia.org/wiki/Cross_entropy

[17] https://en.wikipedia.org/wiki/Triplet_loss

learning rate, enough evidence regarding the learning rates and times should have built up.

The secondary objective was to determine whether the simple models could complete the training test in less than 1 day. The advanced models will only increase or even multiply this base time for training. If a simple model cannot finish its task then the dimension pair will be marked with low feasibility.

As can be seen **(SMTT)[5-6]**, in some cases the training times were getting to pretty high magnitudes, and the consequence was that some of the higher dimensional tests were stopped prematurely.

These results show that the feasibility of training more advanced models on higher dimensional data will require at least some or several months in the worst case scenario, that taking into consideration the available hardware $\{\mathbf{4}\}$.

Learning

The training tests that ended prematurely **(SMTT)[5-6]** did not achieve good accuracy either when compared to the other ones. As a result they were not mentioned due to insignificance.

The first models that can classify fruits have been made **(SMTT)[1,2,4]**. These results prove that even simple models can learn to classify some of the fruits. The accuracy scores, while not being very high, when compared to the random guess with the accuracy of 0.38% seem pretty good.

Forced Overfitting

After learning that the simple models can learn to classify some of the fruits, a simple way of verifying if they can be improved is to force the overfitting on the training data. This process will also prove that there are CNN models that can fully learn to identify each fruit in the dataset.

The target dimension for the experiment will be the average between the remaining successfully tested ones, meaning 26 x 32. This will also reinforce the fact that the dataset is effective even at lower dimensions.

The test **(SMTT)[3]** was successful and the dataset has been almost completely learned by the model.

3.2.3.3. Conclusions

The time tests have proven that training on higher dimension will take months and wont be feasible for the duration of the project. As a result the model development will stop on the cases with feasibility problems.

The successful overfitting and learning tests proved that models, which are able classify the whole dataset, can be built, even on the lower end of dimension size pairs.

3.2.4. CNN Techniques

To improve the performance of models during training certain techniques can be used. For obtaining a better end result, a lot of tests, each with unique modification to the model's structure, will be conducted. By the end of this phase some techniques are sure to prove to be generally good to use during training.

All of the conducted experiments with different techniques will be done on both 13 *x* 16 and 52 *x* 64 dimensions standards. This will help in filtering out which methods are better and usable in different contexts.

The experiments will be done using the model provided above **(Simple Model)** with a numerical increase to the amount of trainable parameters.

Input Image Values

Image data can be represented in multiple formats, such as RGB, HSV etc., but each format usually is composed of an array of integers. The problem that derives from this is having high values as input for the neural network. For a more balanced and efficient training to take hold, usually weights should be encouraged to be as small as possible **[12]**. To solve the issue all the values from each image will be divided by their maximum possible value. The newly acquired image data values will all be between 0 and 1.

The training session tests proved the efficiency of this method by obtaining a bit better values for accuracy but much faster in terms of times and epochs.

Augmentation

Introducing **random flips** as a layer of augmentation is a very practical step to be taken. The model usually will receive as input different fruits images taken in all sorts of positions. The results from the tests show that this technique helped in reducing overfitting but it increased the epochs necessary to attain a certain high values of accuracy.

A more powerful version of flipping the image is introducing **image rotation**. This will produce much more fruit instances from the same root image. While in theory it excels, this method also introduces more noise, while the flipping does not. This noise caused the training sessions to result in less overfitting but the models achieved far less accuracy even with much more epochs. This method does not seem to be worthy of keeping in a real attempt for achieving a good end product.

The **random brightness** augmentation type increases the values of pixels in random locations, the goal being a slightly modified image. This method helped in reducing the overfitting but also lowered the accuracy ceiling and increased the time and epochs needed during the training.

Image Format

All the dataset images respect the RGB[18] standard. The success of training on combined "Grayscale"[19] and "HSV"[20] obtained during the "Fruits-360" project**[31]** incentivizes tests that include different image formats.

The tests will include the "RGB", "HSV", "Grayscale" and "Yiq"[21] image formats and each combination possible of them. As an example, even training on formats such as "RGB + YIQ" is possible.

While not all combinations had great results in terms of accuracy, some managed to get to the same accuracy of the standard RGB format for training in a smaller time space but with a slight increase to the overfitting factor. Some of these examples include: "RGB + any other format", "HSV + Grayscale + YIQ" and "RGB + Grayscale + any other format".

The "HSV + Grayscale" standard used within the "Fruits-360" project **[31]** lowered the accuracy ceiling by 8% on average while not providing any benefit. This is likely due to the big differences between the two datasets {**3.1.3**}

It is worth mentioning that the combination of "RGB + HSV + Grayscale" managed to outperform the standard "RGB" by approximately 3% in terms of validation accuracy on the 13 x 16 dimension, but only managed to get equal results on any other.

[18] https://en.wikipedia.org/wiki/RGB_color_model

[19] https://en.wikipedia.org/wiki/Grayscale

[20] https://en.wikipedia.org/wiki/HSL_and_HSV

[21] https://en.wikipedia.org/wiki/YIQ

Optimizers

Tests show that different dimensions for image data follow somewhat different rules for choosing a good optimizer for training the model. The experiments have been conducted with different learning rates ranging from 10^{-1} to 10^{-5} and have been left to train up to 100 epochs.

The 13 *x* 16 dimension standard seems to achieve similary good results when training models with the Adam[22], RMSProp[23], Adamax[24] and Nadam[25] as optimizers. The other optimizers, namely Adadelta[26] and SGD[27], did not manage to train the model at all.

On the highest tested dimension, 52 *x* 64, only the Adam and Adamax optimizers performed well, while the RMSProp managed to train the model but to a very limited and non feasible extent. The other ones, Nadam, Adadelta and SGD, totally failed to raise the accuracy of the model.

Special Layers

The **fully connected layers**, also named Dense, can be found at the end of the convolutional model **(Simple Model)** and they are responsible for classifying the image features resulted from the convolutional layers.

While in general there is no exact algorithm to determine how many stacked Dense layers result in the most accuracy, experiments with different amount of layers can give some insight to solve the problem **[12]**.

Tests have been done with numbers of dense layers ranging from 2 to 4. On all dimensions standards, having 3 final dense layers with many units has proved to lead to a considerable amount of increased final accuracy, around 5% for the highest dimension and 2-3% for the lower ones. The other test cases had their final accuracy lower or the around the same with the 3 layered experiment.

Dropout layers are usually used together with augmentation **[12]** and their effect on the network is deactivating some of the neurons/units when training, the amount being a percentage given as input. Generally they help in producing a less overfitted model but makes the accuracy fluctuate much more than usual **(T.5)**.

[22] https://keras.io/api/optimizers/adam/
[23] https://keras.io/api/optimizers/rmsprop/
[24] https://keras.io/api/optimizers/adamax/
[25] https://keras.io/api/optimizers/Nadam/
[26] https://keras.io/api/optimizers/adadelta/
[27] https://keras.io/api/optimizers/sgd/

Layer Regularizers

Some tests have been done using the ”L1” and ”L2”[28] layer weight regularizers but the result features an increase in time needed for training to achieve the same accuracy ceilings. In the long run, chances are that regularizers will lead to an increase in accuracy by punishing high values for the model weights. The final conclusion is that the extra time added makes them non feasible in the current situation.

Padding

Due to how convolutions with stride work, the output image of such layers is reduced by some amount, depending on the stride value and kernel size, in each width and height.

As it was calculated during the resizing of data {**3.2.2.2**}, for the pooling process to work accordingly and not lose information from the image, the convolutional layers must not change the shape of the input.

To achieve this, the input for a convolutional layer will be padded before feature extraction such that the output shape is the same as th input one.

3.2.5. Fruits 360 - Model Architecture

To produce more statistics that can help in distinguishing between the ”Fruits-360” **[29]** and ”Fruits-262” datasets, one can compare already built model architectures. One good architecture for comparison is the one from the paper that comes with the ”Fruits-360” dataset, namely ”Fruit Recognition from images using deep learning” **[31]**.

Description

As presented in the paper the training pipeline **[31]** is broken into augmentation and the trainable model. The amount of trainable weights generated by the structure is around 2 million.

The augmentation consists in adding random amounts of saturation and of changes to hue of the image, and also transforming the ”RGB” image pixel data into ”HSV” + ”Grayscale” image data. This process modifies the number of color channels from 3 to 4, implying an increase in information the model has to work with. **(FPTP)**

The convolutional layers all have 5 *x* 5 as filter size, implying that fruit data can contain features not coverable by the general 3 *x* 3 kernels. The pooling layers are standard max polling

[28] https://keras.io/api/layers/regularizers/

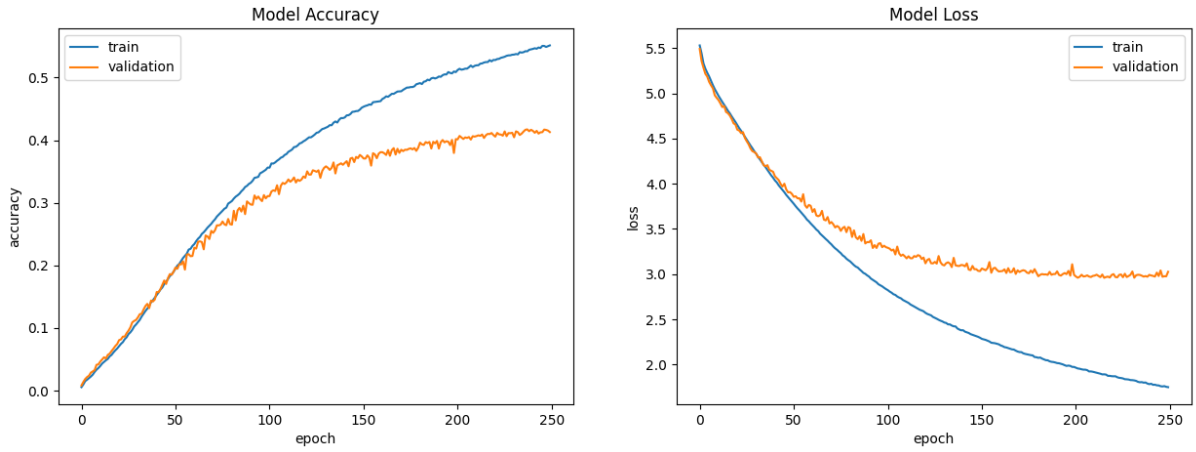
Fruits-360 Paper Training Pipeline + some additions (FPTP)				
Units	**Layer**	**Kernel Size**	**Activation**	**Additional Info**
-	random_saturation	-	-	*flip_left_right
-	random_hue	-	-	*flip_up_down
-	hsv_and_grayscale	-	-	*removed
-	rescale	-	-	scaling the pixels by /255
16 filters	Convolutional2D	5x5	Relu	with padding
-	MaxPooling2D	2x2	-	-
32 filters	Convolutional2D	5x5	Relu	with padding
-	MaxPooling2D	2x2	-	-
64 filters	Convolutional2D	5x5	Relu	with padding
-	MaxPooling2D	2x2	-	-
128 filters	Convolutional2D	5x5	Relu	with padding
-	MaxPooling2D	2x2	-	*only in the improved version
256 filters	Convolutional2D	3x3	Relu	with padding * only in the improved version
-	Flatten	-	-	-
1024	Dense	-	Relu	-
-	Dropout	-	-	20% rate
252	Dense	-	Relu	* the improved version has 1024 units
-	Dropout	-	-	20% rate
262	Dense	-	Softmax	output

layers with 2 *x* 2 layers. **(FPTP)**

For training the optimizer used is the Adadelta[29] and the loss function is categorical crossentropy.

The best session of training, by utilizing the learning rate of 0.1 over 100 epochs, has obtained a model with 41.29% validation accuracy and 60% training accuracy.

[29]https://keras.io/api/optimizers/adadelta/



T.1 - FPTP Model - Training Session

Improved Version

The improved version has at its base the original structure presented but features more trainable weights in the final dense layers. The new amount of trainable weights is around 5 million.

The original model has the output shape 4 *x* 3 *x* 3 after the convolutional layers. Because of this another convolutional and pooling layers pair can be fit to profit from the spatial information.

Noticing that in the case of this project's dataset the Adam optimizer had better success, the training optimizer for the improved version will be this one.

After some tests with the newly improved version, it could be concluded that changing the channels of color for the input did not have any impact on the end accuracy. The results also show that changing the hue or saturation of the images dramatically affected {**3.2.4**} the efficiency of training. As a result, those layers of augmentation have been replaced by image flipping, a better way of augmenting cosidering the current dataset.

The best training session for the improved version features a training accuracy of 75% and a validation accuracy of 53.86%. The result has been obtained by using a learning rate of 0.00025 during 100 epochs in the time span of 1 day.

Conclusions

The default structure used for the "Fruits-360" is most likely built and fine tuned for achieving high accuracy on the original dataset. The dataset presented in this project is different {**3.1.3.13**} and for sure it will require another type of model structure for a better end result.

The improved version is obtained by merely applying some of the CNN techniques that

had success during the testing of different methods {**3.2.4**}. As it appears, some of the better found techniques might prove to be generally good. This will aid in building or improving future model attempts.

3.2.6. Popular Model Architectures

Instead of focusing on developing a model structure from the ground up, a good idea might be to investigate already successful and popular architectures. If any better model structures are found through tests then they could be built upon and fine tuned to work better in this dataset's circumstances.

The model training presented in this phase of the project will be using the CNN techniques that proved to be efficient {**3.2.4**}.

Most of the popular model architectures are built to train on much higher dimensions than 52 *x* 64. As a consequence, some layers of pooling and convolution will be removed in order to scale down the model to a trainable and feasible structure.

VGG

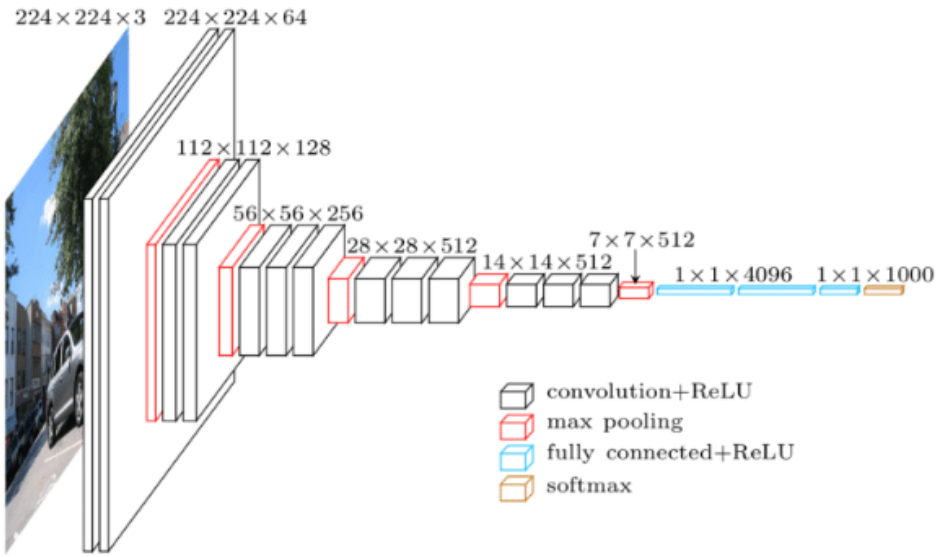
The VGG **[35]** archetype focuses on breaking large scale image classification problems by stacking multiple filters of small size, 3 *x* 3, and using big numbers for the units, thus obtaining very high depth and complexity model.

Both the VGG16 **(Q.1)** and VGG19 **(Q.2)** architectures are individual applications of the VGG archetype, that focused in solving the ImageNet[30] classification task.

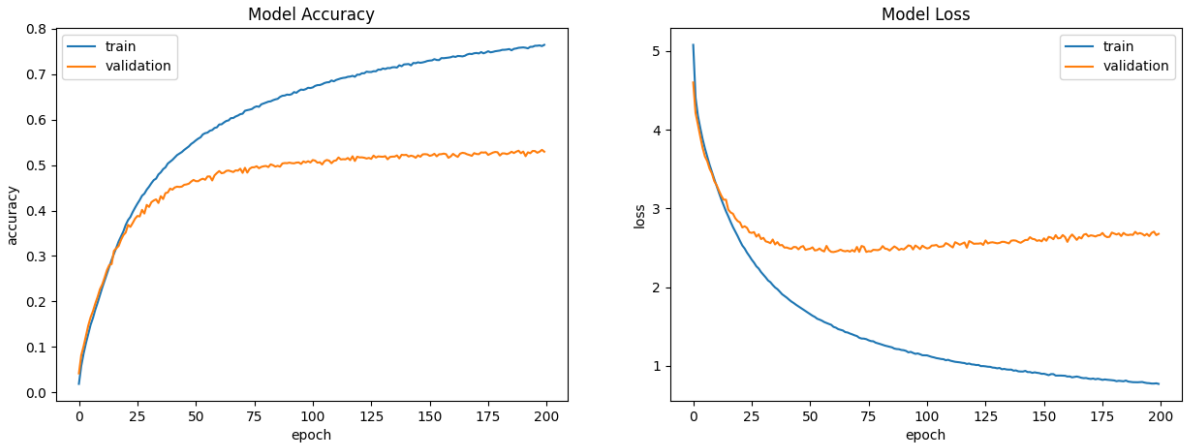
Considering that the current dataset is sparse and requires complex networks to achieve good models, both of the above mentioned structures will be tested.

Considering the advanced depth provided by the archetype, a slower learning approach was used, implying a combination of a lower rate of learning over a longer period of time, 150-200 epochs. Both model attempts were trained using Adam as the optimizer and ended up with around 52% validation and 80% training accuracy **(T.2)** **(T.3)**.

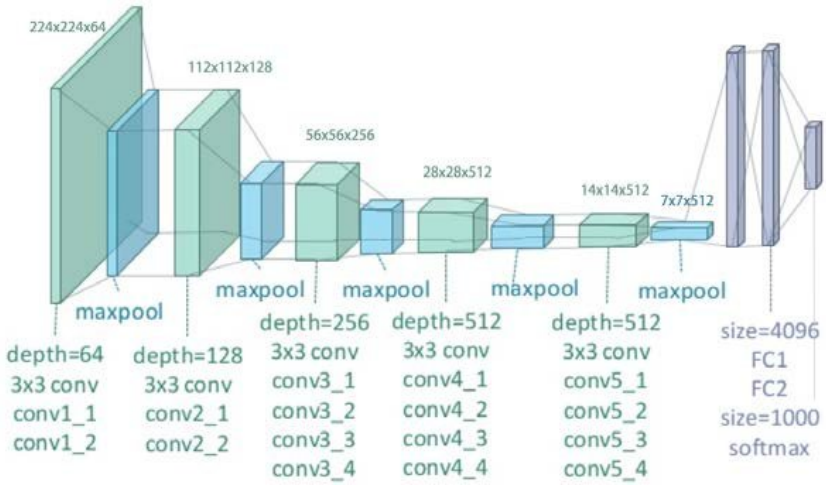
[30] https://www.image-net.org/



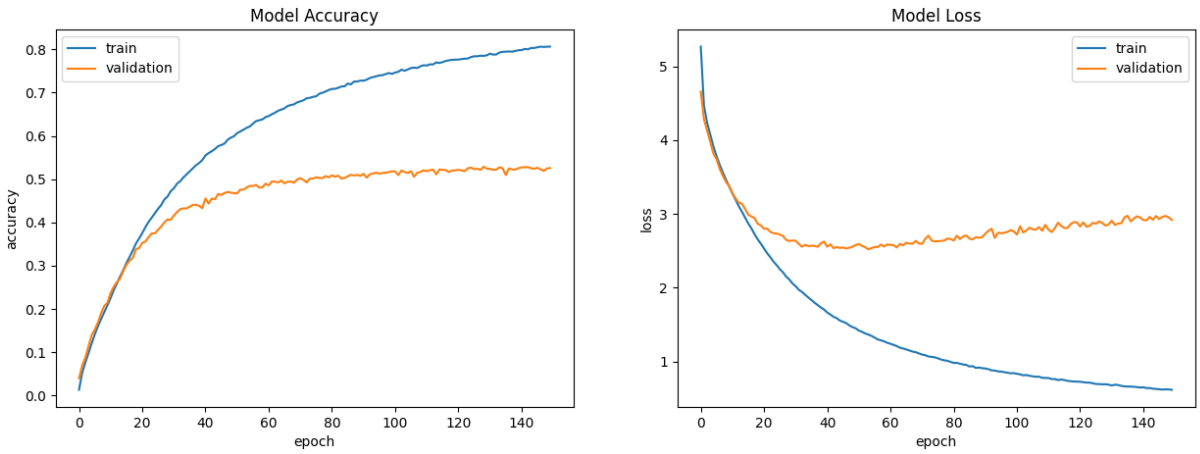
Q.1 - VGG16 Architecture [38]



T.2 - VGG16 Model Type - Training Session



Q.2 - VGG19 Architecture [39]



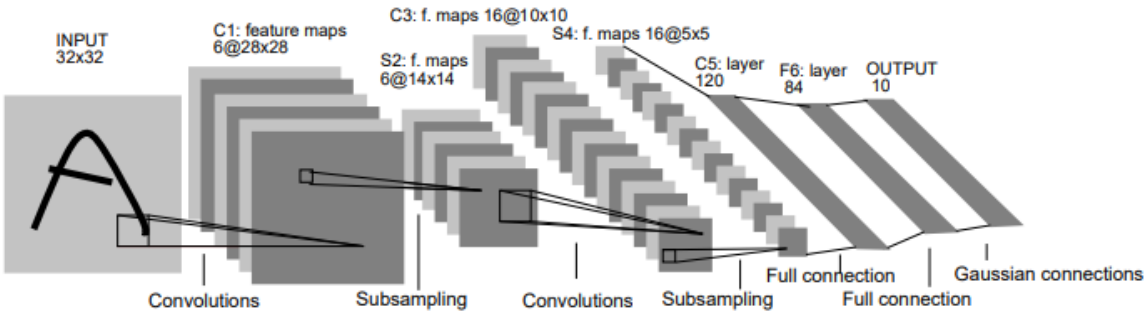
T.3 - VGG19 Model Type - Training Session

LeNet5

The LeNet5 architecture **[36]** is similar to the simple standard CNN architecture, which focuses on stacking convolutions and pooling layers. The main differences consists in using TanH[31] instead of Relu as the activation functions and the convolution layers which use much larger filter sizes initially, and then scalling them down until the end of the model **(Q.3)**.

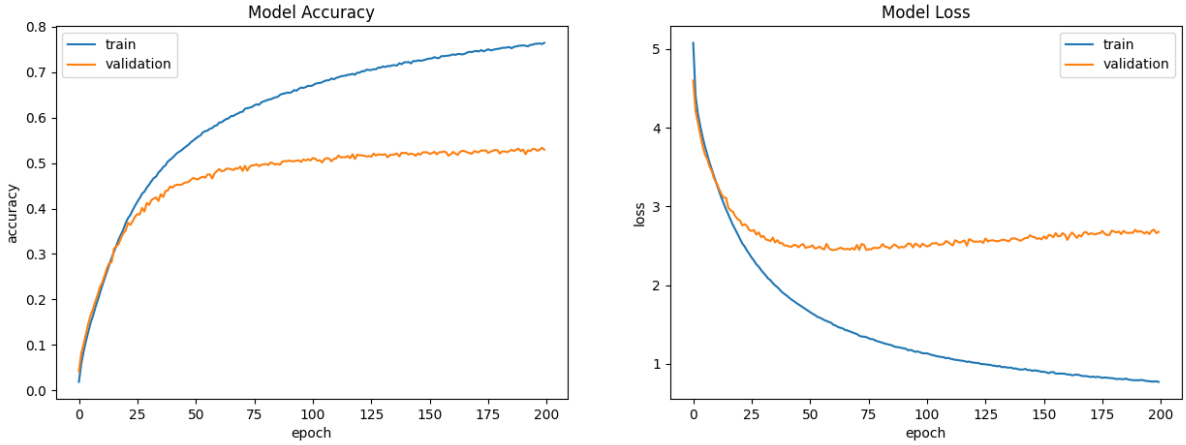
This architecture was one of the first CNN popular architectures and its primary use was distinguishing between different alphabet characters **[36]**. The main advantages of choosing this structure for the model include its simplicity and frequent use which makes easy to compare statistics and also provides useful information about the task.

The model originally **[36]** had around 60 thousand trainable parameters. For the purpose of this project, the number of units have been considerably increased so the model has around 5 million weights and another convolution with pooling layer has been added. As before the training **(T.4)** was done using Adam as the optimizer and the result consists in a model with 51% validation and around 80% training accuracy.



Q.3 - LeNet5 Architecture [36]

[31] https://keras.io/api/layers/activations/



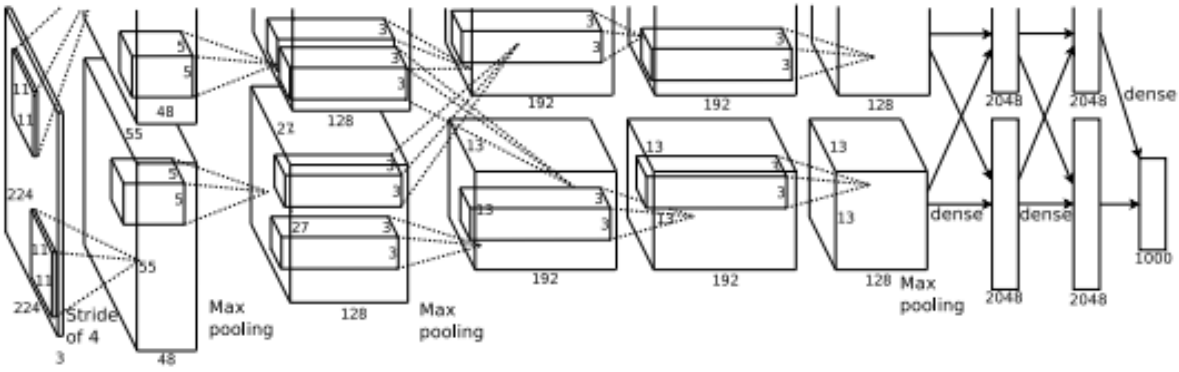
T.4 - LeNet5 Model Type - Training Session

AlexNet

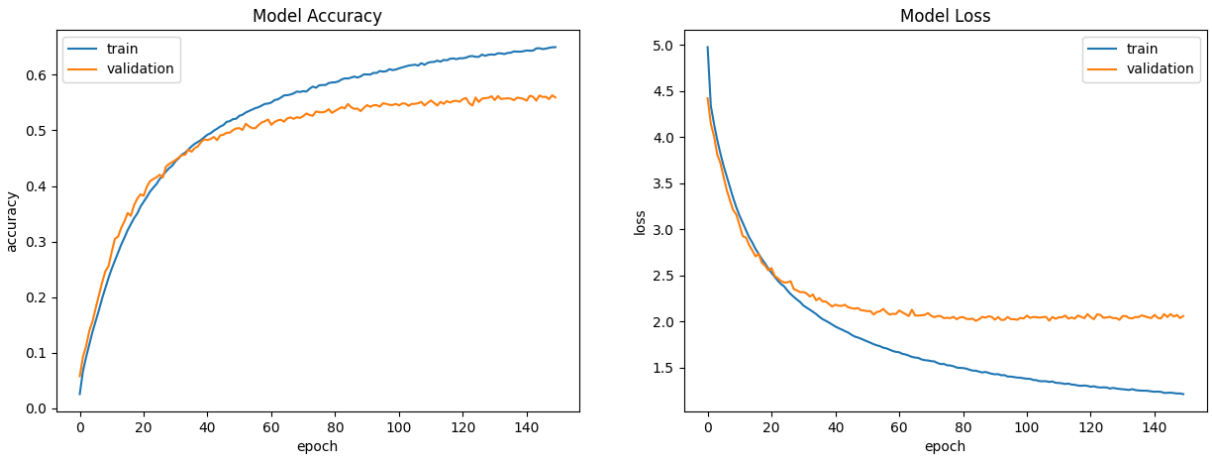
The AlexNet is, like the VGG, a CNN architecture which tried to solve the task proposed by the ImageNet dataset. The structure **(Q.3)** is comparable with the one of LeNet5 **(Q.4)**: it contains multiple layers of convolution with initially big filters that scale down later in the model.

The key difference comes from the Dropout layers added at the end, with a rate of 0.5, which allow for a better model performance, but also allows for training on two separate GPU. Another one of the big changes consists in the structure's scale, the model containing over 10 million trainable parameters and many more convolution layers.

Since the current conditions does not allow for multiple GPU training {**4**}, the training session was slower than it should be, but the final results were pretty good **(T.5)**, arriving at 55% validation and 70% training accuracy in the end.



Q.4 - AlexNet Architecture [37]



T.5 - AlexNet Model Type - Training Session

3.2.7. Final Training Pipeline

After many experiments, the time has come to build the final training pipeline, that will hopefully arrive at the best accuracy.

The final model will receive as input 52 *x* 64 images, hence the highest feasible trainable dimension standard {**3.2.3**}.

Crafting & Training

The base for the model structure will be the previously built AlexNet structure **(FTP)**, hence it proven as the best structure tested {**3.2.6**}.

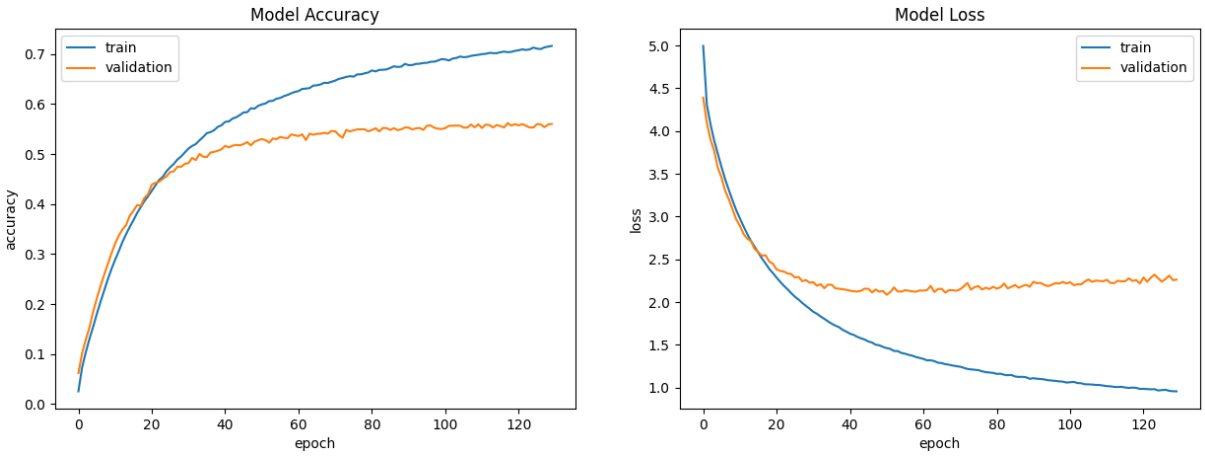
The data will be augmented by flipping it with a chance horizontally and vertically, as it proved the most efective {**3.2.4**}. As an experiment two models will be trained: one with the standard RGB as input and the other with the RGB + HSV + Grayscale.

The convolutional layers will be organized to resemble the ones from the AlexNet structure. Different kernel sizes have been tested but the end results is presented in the final pipeline **(FTP)**. The amount of filters is as high as it could be, but still trainable in a reasonable amount of time, the upper limit being established in one week.

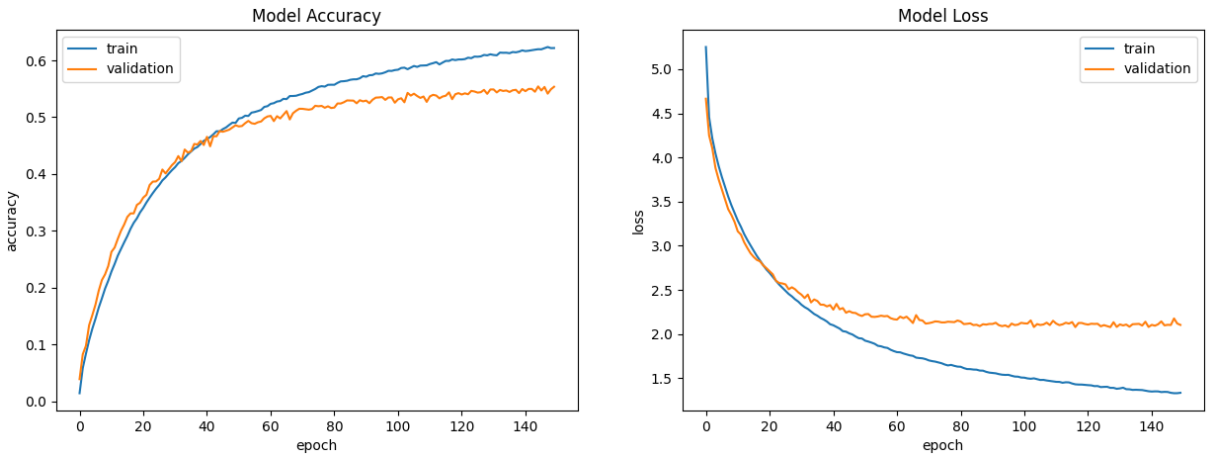
The output of the final convolution layer will be as 4 *x* 2 *x* $color\_channels$. This is due to the combination of pooling layers with convolution, with or without padding added to them **(FTP)**.

The final fully connected layers are in number of 3, one from which is the output, because of the tests conducted before {**3.2.4**}. By the same logic regarding time as before, those layers have as many neurons as possible. The final count for the trainable parameters is around 5 million.

Final Training Pipeline (FTP)				
Units	**Layer**	**Kernel Size**	**Activation**	**Additional Info**
-	flip_left_right	-	-	-
-	flip_up_down	-	-	-
-	convert to Gray + HSV + RGB	-	-	*only second model
-	rescale	-	-	scale the pixels by /255
64	Convolutional2D	11x11	Relu	padding
-	MaxPooling2D	2x2	-	-
128	Convolutional2D	5x5	Relu	padding
-	MaxPooling2D	2x2	-	-
256	Convolutional2D	3x3	Relu	-
256	Convolutional2D	3x3	Relu	-
256	Convolutional2D	3x3	Relu	-
-	MaxPooling2D	2x2	-	-
-	Flatten	-	-	-
-	Dropout	-	-	50% rate
1024	Dense	-	Relu	-
-	Dropout	-	-	50% rate
1024	Dense	-	Relu	-
262	Dense	-	Softmax	output



T.6 - (FTP) - First - Training Session



T.7 - (FTP) - Second Model - Training Session

The training session has been done with a learning rate of 25 x 10^{-5} in batches of 256 images. The RGB model took around 130 epochs to reach the ceiling for accuracy (57% validation accuracy, 75% training accuracy) and the RGB + Grayscale + HSV model peaked in 150 epochs (56% validation accuracy, 66% training accuracy).

Tests & Conclusions

As it can be seen (**TDR**), the best models that have been obtained so far have been tested on the test data prepared before hand {**3.2.2.1**}.

Due to the nature of augmentation layers which make the model output non deterministic, each image from the test data has been put through the classification pipeline 1000 times for more generalised results. The results from the table show the global average prediction accuracy that has been obtained on all the inputs.

The 13 x 16 model that was used is the best one resulted from testing various techniques before {**3.2.4**}. Even in an not optimised state, this model obtained pretty high scores of accuracy when compared to the random chance of 0.0038%.

The 52 x 64 models are the result from the previous section, and their results are at the same time pretty similar and with a high accuracy for top predictions.

3.2.8. Image Classification Pipeline

Having obtained the final results for the models, the pipeline, which takes an image as input and produces predictions based on it, can now be built.

The pipeline begins with loading a specific model which will be used for predictions, this step also fixes the input dimension for the image.

Final Models - Test Data Results (TDR)			
Model	**Top 1 Prediction Accuracy**	**Top 5 Predictions Accuracy**	**Top 10 Predictions Accuracy**
13x16 the best test model obtained	42.47%	58.57%	65.88%
52x64 RGB	59.15%	80.4%	86.66%
52x64 Grayscale + RGB + HSV	58%	79.24%	85.69%

The next step is loading a random image or a set of images and resizing them **(D.3)** to fit the input of the model, the paradigm being same as before {**3.2.2**}.

Due to the nature of the models, which will randomly flip the images horizontally and vertically, each image will be put through the model a noticeable amount of times so the results become stable and meaningful, usually around 1000 times.

After obtaining all the result vectors for one image, they will be summed up and divided by how many times that image went through the model. The result will be and array with the average predictions estimated in percentages, which can now be sorted to obtain the top most predicted labels. The amount of predictions shown depends on the use case, which might make it is useful to know more than just the first prediction **(D.3)**.

The results for the test data **(TDR)** have also been obtained using the pipeline described above.

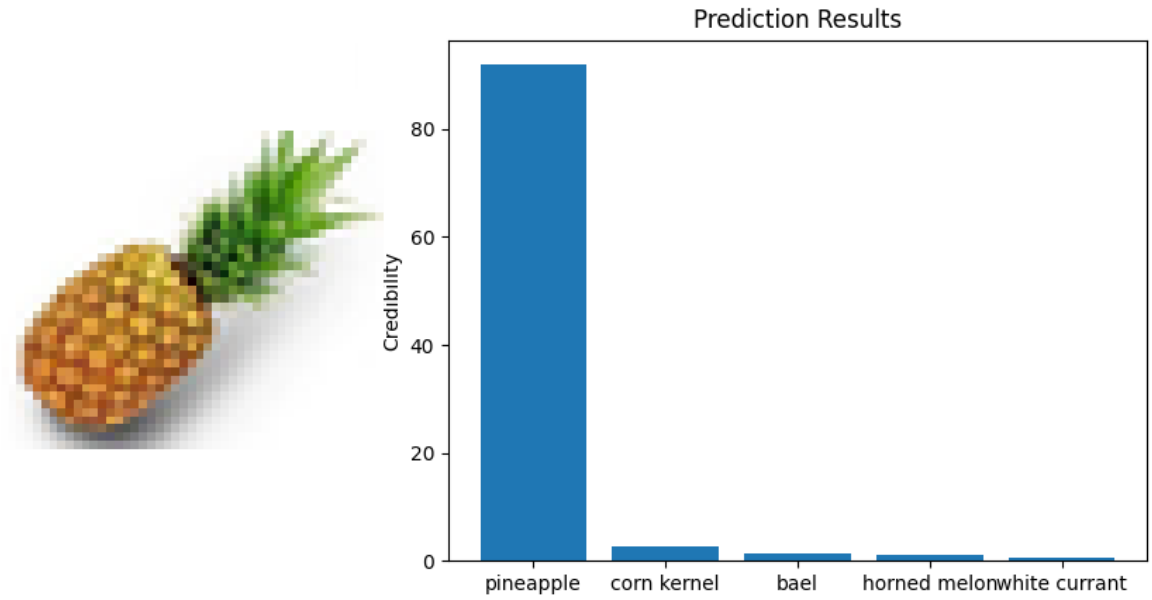
3.2.9. Problems

Dataset Imbalance

The decision to train on the imbalanced dataset and having labels with under 100 specimens and others with over 1000 definitely has negatively impacted the end results. In further experiments, labels with small numbers of images could be eliminated or integrated in similar fruit families.

Indistinguishable Fruits

One problem discovered during the final stages of the project, specifically when building the prediction pipeline, is that some of the ”berry” fruit types tend to be very similar; to the



D.3 - Pineapple Image - Top 5 Predictions

point that even the human eye has trouble distinguishing between them **(D.4)**.

D.4 - Hackberry vs Blueberry

The example presented **(D.4)** is not the only one, and almost all of fruits that fall under this category fail to find the correct prediction in the first try.

Loading Times

One big problem, encountered while testing the feasibility of the simple models {**3.2.3**}, is that loading a lot of data at once from the storage device {**4**} can take a long time and in turn make training a model very slow.

The solution concluded in storing the dataset in a much streamlined way, which allowed for fasterloading and no time wasted on changing the structure of information.

Model Architecture

Choosing the perfect model architecture for a problem that falls under the domain of neural networks is very difficult and becomes even more complex with lesser experience. The scarcity of similar tasks faced by CNN models also adds more complexity to choosing the right structure.

While the final models are more effective {**3.2.7-8**} than the random guess, the method for building them is not. Choosing somewhat randomly what tests to make and which tests actually provide useful information increases the time needed to make an efficient model.

3.2.10. Conclusions & Future Work

Time & Better Planning

Working with different architectures and pipelines for training models proves empirically that most of the compromises that will be done, when building a CNN structure, are between accuracy and time.

Having limited amounts of time can hurt the final product. The goal from the beginning was to explore many architectures and possibilities for training the model and therefore for the final model structure some parameters have been tweaked for it to have a feasible training session.

The next iterations of the project will most likely sacrifice more time researching and planning tests and model architectures. Having results that come from much less chaotic experiment chains will most likely conclude in better models with higher accuracy.

Techniques

There are many techniques {**2.3.3**} that have not been tried and many others that were not researched enough. Seeing that the simple model can overfit and learn the entire dataset even on the 26 *x* 32 dimension standard {**3.2.3**} only proves that the final model is very far from the best achitecture.

Eliminating or merging certain labels could increase the accuracy of the final product and provide more insightful information when trying different experiments with the dataset.

Final Result

The accuracy of the final models can be considered pretty low, especially when one would like to use them for real applications. Analyzing the statistics for the top 5-10 predictions **(TRD)** and some of the individual fruit image examples **(D.3)**, one can realize that an application based on those models will manage to find the fruit in the top 5 predictions most of the time, which makes it usable but not for serious botanical problems.

4. Hardware & Specifications

The times, accuracy scores and even some algorithms couldn't have been ran or obtained without the use of certain pieces of hardware which makes this section all the more important for the results that have been obtained.

All the processes and algorithms utilized during the building of the thesis were run on the same machine with Windows 10 Home (latest version as of 2020-2021) as its main OS.

Table H	
Component	**Specifics**
CPU	**AMD Ryzen 7 2700X ∼ 8 Cores (16CPUS) ∼ 4.0GHz**
GPU	**AMD Radeon RX 5700 XT SE ∼ 8GB VRAM ∼ 1860 MHz**
RAM	**16 GB DDR4 ∼ 3200MHz**
Storage	**Seagate FireCuda 2TB SSHD SATA-III 7200RPM**
Motherboard	**Asus Rog Strix X470-F**
Source	**680W**

An important aspect about the build provided **(Table H)** is that the AMD GPU and CPU make use of the memory sharing technology **[32]** that AMD provides which directly impacts CNN training speeds on the GPU utilizing the direct-ml[32] library or other operations on the GPU. Usually this optimization is available on CPUs that are from the 5000 series or newer but ASUS has provided a way to make use of this technology from the 1000 series onward.

The time needed for training the most complex of models built within this project was at most 7 days of continuous running, hence it cannot be guaranteed that all the theory crafted models, that were not trained, can be trained on the machine. Other algorithms or scripts did not take any considerable amount of time in comparison to the machine learning ones.

At this moment the only library for python on windows that allows CNN training on an AMD GPU[33] is direct-ml and as a consequence the whole training has been done utilizing this module on Python3.7, with all of the other libraries down versioned also. **[33]**

[32]https://pypi.org/project/tensorflow-directml/

[33]https://developer.amd.com/resources/developer-guides-manuals/

5. Final Conclusions

4.1. Conclusions

The project has started with the goal of having a dataset for plants on which to build CNN models for classification. The original plan had many flaws which gradually showed up throughout the time space. In the end, the project managed to generate the ”Fruits-262” dataset **[28]** from fruit images found on the public domain, which also led to some averagely good CNN models.

Dataset

Due to feasibility, the problem of building a plant dataset has transitioned to building fruit based one. After various stages of automatic and manual filtration the dataset has arrived at its final iteration, covering 262 fruit species with over 226 thousand images.

Considering other fruit datasets, like ”Fruits-360” **[29]**, the collection of data, built during this project, is much more sparse but is very unbalanced and tends to have images with high degree of similarity.

Models

The ”Fruits-262” dataset, while being organized, was not in a very efficient state for training CNN models on. Converting the data into arrays with image data and their labels, all serialized on the storage device, made the training sessions much more efficient **(Table C)**.

For building the better final models, which have around 58% test accuracy, a lot of initial testing and exploring have been done. Data augmentation and the Dropout Layers pushed the models to higher plateau of accuracy; and , together, with the structure of AlexNets and a good amount of trainable weights have obtained the peak results.

While testing the final models by using as input individual fruit images, another problem arose, one which involves various species of berries being hard to distinguish. It turns out that some berries **(D.4)** are hard to compare when they are allowed to be in peculiar positions.

4.2. Future Work

Similarity

Having images with a high degree of similarity, under the same label type, leads to a higher end model accuracy since they can be treated as approximate duplicates but negatively impacts how the model performs in real scenarios.

Having an increased variety for a fruit label improves the usability and stability of the developed models. A further iteration of the dataset could remove this problem by using the said instances of images and transform them with some method of augmentation.

Imbalance

The dataset contains some labels which, by comparison, do not contain high volumes of data. This introduces a high degree of discrepancy between fruits during the training of the model, which further impacts how well the model will classify them.

Fixing this problem would either require the removal of the said labels, which is easy to do but not very elegant, or find a method to increase the scarce quantity of information. The method that will be used for obtaining more data will most likely be one already researched within the project {**3.1.3**}.

Indistinguishable Fruits

Since the latter model individual tests have proven the fact that some fruit species are hard to separate, either the dataset or the model training pipeline will have to go through different changes.

One method for solving the problem is having the dataset merge the labels with this problem, thus creating a label family. The resulting families will have separate tasks of classification attributed to them, the principle of **divide et impera** being the key of this process.

The other option is to find or create a model architecture capable of separating the features of the said fruits with a much higher accuracy. This approach can be done with direct changes to the Convolutional Structure, the augmentation layers, by applying a new type of CNN technique or a combination of all the above mentioned.

4.3. Appreciation

For projects on the scale of Bachelor Thesis to be built, many people need to get involved and help. Without help, guidance and research coming from other people none of this thesis could have been possible.

First and foremost I would like to thank **Prof. Dr. Adrian Iftene** for guiding and helping me through such a massive scale project. From the initial results to the writing of this chapter, I learned many things and took various dangerous decisions, all of which could not have been possible without his support.

I respect and appreciate very much the help of my friend and colleagues that work on similar subjects: **Crețu Bogdan Antonio** and **Ștefan Pâsoi**. We all have faced similar problems, maybe in different contexs and circumstances, and while the scale of our separate projects managed to continuously grow, we still found the time to help and guide each other, fact for which I am very thankful for.

I would like to thank two of my professors: **Samson Mihai** and **Valentin Roșca**, for providing additional support and guidance during very difficult phases of the project.

The process of manual filtering of the dataset required large amounts of work and time, for which many of friends offered to help. Everyone has given some of his or hers individual time to help in building what is called today the ”Fruits-262” dataset. For this and their desire to help me through this journey I would like to thank **Minuț Emilia-Georgia**, **Alexandru Dumitriu**, **Profir Petrișor**, **Loghin Vlad** and **Onu Robert**.

Finally, I would like to thank **my family** and all **my friends** for directly, or indirectly, supporting me in many different, yet very meaningful, ways during the course of this project.

Inspiration and brilliant ideas come from the people constantly surrounding us, even if we do not manage to notice or locate the exact origin. As a result I would like to thank **everyone**, from the **”University of Alexandru Ioan Cuza - Faculty of Computer Science”** from **Iași**, that went through actions, altering the timeline, that lead to this very moment.

6. Bibliography

1. Aakash Moghariya - 2017 - Popularity of CNNs in computer vision tasks. Quora.com

2. Shuai Wang, Zhendong Su (2019) Metamorphic Testing for Object Detection Systems. https://arxiv.org/abs/1912.12162

3. Mark Maloof (2017) Artificial Intelligence: An Introduction. Department of Computer Science Georgetown University, Washington, DC 20057-1232, http://www.cs.georgetown.edu/ maloof

4. Stuart Russell, Peter Norvig (2020) Artificial Intelligence: A Modern Approach. http://people.eecs.be sell/intro.html

5. T. S. Huang (1996) Computer Vision: Evolution and Promise. University of Illinois at Urbana-Champaign Urbana, IL 61801, U. S. A.

6. Tim Morris (2004) Computer Vision and Image Processing. In Cornerstones of Computing. Red Globe Press.

7. Xin Fengab, Youni Jianga, Xuejiao Yanga, Ming Duc, Xin Lib (2019) Computer vision algorithms and hardware implementations: A survey. In Integration, Volume 69, November 2019, Pages 309-320.

8. Moore, Gordon E. (1965) Cramming more components onto integrated circuits. In Electronics, Volume 38, Number 8, April 19, 1965.

9. Torsten Fink, Michael M. Gutzmann,Ralph Weper (1998) Advances in Parallel Computing. Parallel Computing Fundamentals, Applications and New Directions, Edited by E.H. D'Hollander, F.J. Peters, G.R. Joubert, U. Trottenberg, R. Völpel, Volume 12, Pages 3-748.

10. Tom Mitchell (1997) Machine Learning. In McGraw-Hill Science/Engineering/Math; (March 1, 1997).

11. Ian Goodfellow, Yoshua Bengio, Aaron Courville (2016) Deep Learning. MIT Press.

12. David Stutz (2014) Understanding Convolutional Neural Networks. Fakultät für Mathematik, Informatik und Naturwissenschaften Lehr- und Forschungsgebiet Informatik VIII, Computer Vision.

13. Chris Snijders, Uwe Matzat, Ulf-Dietrich Reips (2012) "Big Data": Big Gaps of Knowledge in the Field of Internet Science. In International Journal of Internet Science, Volume 7, Issue 1, pages 1-5.

14. Rashi Desai - Towards Data Science - 2020 - How to build your own dataset for Data Science projects

15. Alexandre Gonfalonieri - Towards Data Science - 2019 - How to Build A Data Set For

Your Machine Learning Project

16. Wikipedia - 2020 - List of All Plants

17. Half Your Plate - 2020 - Fruits

18. Wikipedia - 2020 - List of All Fruits

19. List Challenges - 2020 - Every Fruit in the World

20. Fruitsinfo - 2020 - List of All Fruits

21. WebArchive - Facts about Google and Competition

22. Valohai - 2019 - Copyright Laws and Machine Learning

23. Boston College - 2020 - Copyright and Fair Use

24. Creative Commons - CC0

25. Swadhin Agrawal - 2021 - 13 Best Image Search Engines

26. Selenium - Python3 Module

27. Google - 2020 - ReCaptcha System

28. Minut Mihai Dimitrie (Aelchim Minut) - 2021 - Fruits-262 Dataset

29. Mihai Oltean - 2018-2021 - Fruits 360 Dataset

30. Rafet Can Kandar - 2021 - Image Classification Using CNN - Fruits

31. Mihai Oltean & Horea Mureșan - June 2018 - Fruit Recognition from images using deep learning

32. AMD - 2020 - Smart Access Memory

33. Microsoft - 2021 - Direct ML

34. Justin Ruan - 2021 - Fruit 360 using Vision Transformer

35. Karen Simonyan & Andrew Zisserman - 10 April 2015 - Very Deep Convolutional Networks for Large Scale Image Recognition - arXiv:1409.1556v6

36. Yann LeCun, Leon Bottou, Yoshua Bengio and Patrick Haffner - November 1998 - Gradient-Based Learning Applied to Document Recognition - lecun-98

37. Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton

38. T. L. I. Sugata & C. K. Yang - November 2017 - VGG16 Architecture Image

39. Clifford K. Yang & Yufeng Zheng - May 2018 - VGG19 Architecture Image