Fall 2020

Using Object Detection Algorithm and Optical Character Recognition to Read Data from alphanumeric tags in text

Ana Bazerque

Davi Moraes

Marcela Souza

CCT, Dublin, 19/05/2020

OCRscience
Image Recognition Software

Using Object Detection Algorithm and Optical Character Recognition to read data from alphanumeric text in tags

Report and Documentation

B.Sc. in Information Technology / Applied Technology Group Project
Developed by: Ana Bazerque, Davi Moraes and Marcela Souza
Supervised by: Mikhail Timofeev

cct

Abstract

The present document explores the use of machine learning techniques, specifically supervised learning and classification. It applies those techniques to create a solution for a real world company that provides medical products and services to hospitals. This project will deal with streamlining the calibration of medical weighing scales. The developed application will use object detection and character recognition to identify and classify a digital image of a scale's tag, and fill in a form with the corresponding data. The main reason for the need of this application is to avoid human errors and automate the collection of data from the scales.

Acknowledgments

The completion of this project would not be possible without the help, the patience, and the collaboration of a few people outside of the group. We would like to thank our supervisor Mikhail Timofeev, for helping us keep our feet on the ground and to not become another Fyre Festival (inside joke). We also need to thank Mr. Matt McParland, from Accuscience, for engaging with us in this journey and providing all the information required to the continuity of the development. Tiarnán McCarville had a priceless participation on this project by taking pictures of the tags during his working hours and that as well needs to be mentioned. There is also Fernando Henrique Molina Ferrarezzi, who provided the group with some valuable technical support during the build of this application, as well as the active online tech community for their dedication to sharing knowledge. We would like to thank our partners, family and friends that were with us during this process.

Table of Contents

Glossary

In this document:

A

AI - Artificial Intelligence
API - Application programming interface
AutoML - Automated Machine Learning

B

Base64 - binary-to-text encoding

C

C - Programming language
CUDA - Programming language
cuDNN - CUDA Deep Neural Network
CNN - Convulucional Neural Network
Colab - Google Colaboratory IDE

D

Darkent - Open source neural network framework

E

F

FPS - Frames Per Second

G

GPU - Graphics Processing Unit

H

HTML - HyperText Markup Language
HTTP - HyperText Transfer Protocol

I

IDE - Integrated Development Environment

J

JSON - JavaScript Object Notation
Jupiter - Server-client application that contains both computer code and rich text elements

K

Keras - Machine learning framework

L

LabelImg - Open Source Labeling Tool
LaTex - Document preparation system

M

mAP - mean Average Precision
ML - Machine Learning

N

NVIDIA - Technology Company

O

OCR - Optical Character Recognition
ODA - Object Detection Algorithm

P

Python - Programming language

Q

R

R-CNN - Region-based Convolutional Neural Networks

S

Seca - Seca Company

T

Tanita - Tanita Company, Medical Scales Manufacture
Tesseract - Open source optical character recognition engine

TPU - Tensor Processing Unit

U

V

W

X

Y
YOLOv3 - You Only Look Once version 3 object detection algorithm

Z

Application's Repository and Other Resources

GitLab Repository https://gitlab.com/madcode2017/ocrscience.git

To run the API:

Install Python 3 any version https://www.python.org/downloads/

Install Tesseract https://github.com/tesseract-ocr/tesseract

From the root folder, type into the command line:

```
1: cd mad_api

2: pip3 install https://github.com/mitsuhiko/flask/tarball/master

3: source env/bin/activate

4: export FLASK_APP=app.py

5: export FLASK_ENV=development

6: pip3 install -r requirements.txt

7: flask run
```

Colab step by step model training

https://colab.research.google.com/drive/1HrZaUK1vh5sxMdwMgd507Qbex20Y833S?usp=sharing

Product Screencast https://youtu.be/sVSVnA4bypA

Introduction

Machine Learning is one of the most interesting subjects to explore nowadays and it is also the core of this project. It all started by identifying a demand, a real world situation to be challenged into an automated solution. This is when Accuscience came into the picture, "Accuscience is a leading provider of Medical, Surgical & Scientific Products & Services to Hospitals, Pharmaceutical Manufacturers and Academia." (Accuscience, 2020) What this project will focus on will be the calibration of medical weighing scales, with the engineers who perform the calibrations being the end users of the application.

This project delves into building an application that will use supervised machine learning to create a model that will make predictions based on a known set of input data and known responses to the data. From this reference, it will train a model to generate sensible predictions in return to new input data through classification, one of the machine learning's approaches. This process will then guarantee the correctness of the input, which is now being produced on a paper form. With this solution, the use of paper sheets will be eliminated from the engineers routine while saving the new image inputs into a database for further training and improvement of the model.

The research conducted will focus more on the practical aspects of the product being developed, rather than on theoretical concepts. Although they are currently in a condensed maner for reference. Because there is a large range of options and also due to the initial inexperience of the group members with the subject, extensive trials were performed until the final solution and combination of technologies were chosen.

Project Overview

Accuscience is an Irish company providing a wide range of medical and laboratory related services including Sales, Marketing, Distribution, and Support. Among the many services provided by Accuscience, the present project is specifically focused on the scales calibration service. To keep records of the inspections made by Accuscience's engineers, the company has a custom system. However, the current system does not handle the scales calibration services.

When a scale calibration is required, the engineers are sent on site. For each customer's location, they need to fill in a form - on paper - for each of the individual calibrations. The form contains the details about the specific equipment, such as serial number, ID number, manufacture, and model. The engineer then puts a sticker on the machine that says it had been successfully calibrated. The sticker identifies the engineer and the day the calibration was fulfilled. The forms are later individually scanned to be digitally recorded. Since some of the locations can have more than 200 scales to be calibrated, the paperwork can become a big hassle.

The time and resources necessary to complete a full cycle of inspection are costly and unproductive. The aim of this project is to implement a digital application that optimises in many ways the entire process of data collecting and recording. By reducing the time spent filling in forms and scanning them, an automated application would also increase the reliability of the inputs from the engineers. The application will substitute the need for a paper form and manual scanning of the documents. It will work using ODA (Object Detection Algorithm) and OCR (Optical Character Recognition) to read the labels on the scales and recognize the serial numbers and any other information necessary. Therefore, after reading the serial number, the application should then populate a digital form with the specifications about the referred scale and enable fields for the input of the test results. Another benefit of an automated system with character recognition is the savings on using no paper sheets and at the same time being more environmentally conscientious.

Product Context

Even though the application will be served by an API and needs connection with the internet, the product will not connect to their current system. The reason for that is that their current system deals with a higher level of precision and credentialing which is not necessary for the scales' calibration. For the moment, the company has one engineer in charge of the scale calibration, with one or two extra engineers in case of a larger demand for inspections, but the application will be used by only one user most of the time.

User Characteristics

The application will have only one user type, which is the engineer who will conduct the inspection and its individual calibrations. They will have access to the application through a mobile device with a camera. The engineer will be responsible for overseeing the specifications of the inspections, making sure the calibrations are handled properly and populating the application with the results. Since they can be sometimes a little negligent while filling in the form, the application needs to guarantee that, e.g. the serial number, which is a long string of digits, is inserted correctly in the form. The problems go from illegible handwriting to missing fields, and it is crucial that the system will take the basic information about the individual scale from its label.

Assumptions

All engineers/users of the system need to carry a device with access to a camera to capture the label. They will need access to the internet. Also, the tags/labels are assumed to be readable.

Constraints

The current conceptual model of the (mechanic) system is going to be kept and used as a reference for the application. A successful implementation of the system into a software version has to consider the amount of time spent by the user to fill in the forms. Using the application should be at least as quick as completing paper reports.

After the conclusion of each calibration process, a PDF version of the form should be generated and saved.

The project also has time constraints, since the new system needs to optimise the time spent on filling forms, which means that the time spent on inputting information into the application should ideally be shorter than doing it handwriting.

Furthermore, the data collected about calibrations should be locally stored in an approachable format which can be easily processed.

Problems with the current system

- The current system relies mostly on paper forms and handwriting.
- The handwritten paper forms are too susceptible to human error.
- The processing of all the paperwork is time consuming.
- The handwritten paper forms have to be manually scanned to the database.

- The Service Report, with a summary of the inspection, has to be manually produced.

Objectives of the new system

- To provide an easy to use application that acts like an interface between the database and the user that will carry the inspection.
- Implement Object Detection to identify the object classes in the images captured with the camera.
- Send the data identified by the Object Detection Algorithm to an OCR (Optical Character Recognition) system to capture serial numbers in the selected areas of the image.
- Optimise forms that will process the data from each calibration and save it into the correct database.
- Supply users with access to the inspection's description and details by syncing the application on the mobile device with a central server.
- Provide a web interface and mobile application.

Scope of the new system

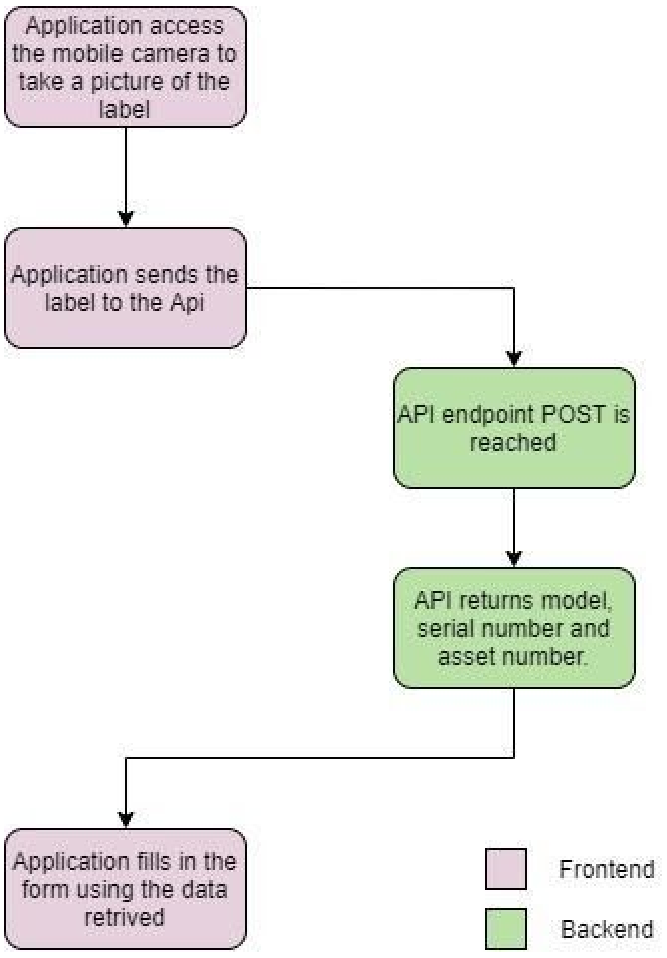
The project will tackle the following processes:

- Handling the calibration of not-precision scales.

- Reading the information from scale’s labels using ODA and OCR.
- Populating the form with the details about that specific scale.
- Populating the form with the reference weights for the scale.
- Inputting the information about the conducted tests into the form.
- Aggregating the details about an inspection in one document with a unique identifier.

Application Flow-Chart

The application should perform following the flow described on the chart below:



Application access the mobile camera to take a picture of the label
Application sends the label to the Api
API endpoint POST is reached
API returns model, serial number and asset number.
Application fills in the form using the data retrived
Frontend
Backend

Image 1 - Flow Chart

In detail, the user will upload a picture of a tag to the system. In the backend, this picture will be processed and compared to the personalised model, which will return cropped images of the found object's bounding boxes. These cropped images are then sent to the Optical Character Recognition algorithm to run the character recognition. This algorithm returns a string, which is parsed into Javascript Object Notation and sent to the frontend to fill in the calibration form with the information corresponding to the specific scale. The ideal outcome is that the application should be able to work without connection to the internet to solve the predictions and upload the image later on to a database for further training of the model.

Project requirements

- The system must be able to recognize characters from a picture of a label using Object and Character Recognition.
- The system needs to be populated with a large library of labelled images in order to collect all the details.
- The form should contain the details about "Reference Weights" that are applied to the equipment.
- The "Reference Weights" should be aggregated in groups.
- The form must contain the details about test points.
- The system must generate a unique ID number for each and every inspection.
- Each inspection can have one or more individual calibrations.
- The system must generate a unique ID number for each and every calibration.

- The details contained in the scale record will be automatically populated into the individual calibration form.
- The system should provide a form for the engineers to add the details of each and every calibration.
- The engineer must add the results about each and every individual calibration to the system.
- The engineer must be able to add the result of the calibrations.
- The engineer must take a picture of the serial number of each and every equipment he verifies.
- The form must contain an option for “Customer not Present”.
- The engineer must be able to include “As found” details.
- The engineer must be able to include “As left” details.
- Numeric fields must allow decimal points.
- Once the inspection is finished, the engineer should submit the information into the system.

Methodologies

Machine Learning

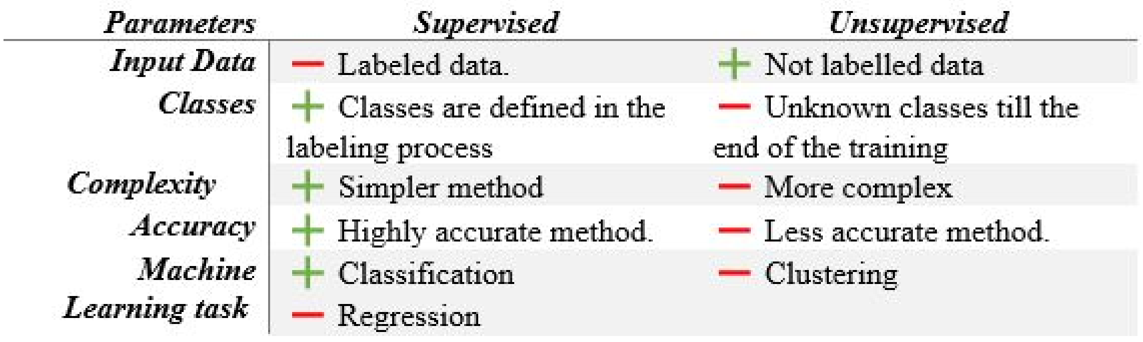
"The term machine learning refers to the automated detection of meaningful patterns in data." (Shalev-Shwartz, and Ben-David, 2014) It is a subfield of Artificial Intelligence (AI), where the machine receives a set of inputs and is expected to learn from it without the need of being specifically programmed for this task.

Convolutional Neural Networks

Neural Network is a machine learning algorithm composed of a large number of simple, connected processing units, each producing a sequence of real-valued activations. Input units are activated through sensors which analyse the environment and other units through connections with real-valued weights from previously active units (Schmidhuber, 2015). In other words, they are computing systems generally based on the biological neural networks that form any animal's brain. Such systems can be taught to perform tasks by considering previously seen examples, generally without being programmed with task-specific rules (Maglogiannis, 2007).

A CNN can be trained with supervised or unsupervised learning. In the supervised learning the model learns from a labeled dataset, that teaches the program the general aspects of what is being asked to look for. This makes the model capable of identifying the previously seen objects that belong to the given classes (Sathya, R. and Abraham, A., 2013). On the other hand, an unsupervised learning will learn from an unlabeled dataset and will cluster the objects according to the found patterns. With this method, the final classes are unknown until the end of the training(Schmidhuber, 2015).

Supervised vs Unsupervised Learning



Parameters	*Supervised*	*Unsupervised*
Input Data	− Labeled data.	+ Not labelled data
Classes	+ Classes are defined in the labeling process	− Unknown classes till the end of the training
Complexity	+ Simpler method	− More complex
Accuracy	+ Highly accurate method.	− Less accurate method.
Machine Learning task	+ Classification − Regression	− Clustering

Table 1 - Comparison between supervised and unsupervised training

Conclusion

The process of labeling the data set for a custom model is a process that needs to be done manually and for this reason it is time consuming and subject to human error. On the other hand, the algorithm involved in supervised learning is simpler because it does not require the model to find unknown patterns. Instead, it will focus on learning the features of a pre-existing class. Taking in consideration both training methods and the requirements of this project, the supervised learning appears to be the most adequate. The reason is its accuracy and specially because supervised learning ability to solve classification problems.

Classification

In this task, the computer program is asked to specify which categories some input belongs to. “The combination of both components, i.e. information extraction and application, is called data classification; in this context, the aim is to develop tools that, having studied a large labeled base of available data, are able to automatically label not previously seen data.”(Maglogiannis, 2007) In this scenario, the task outputs the probability distribution over the preset classes. Object Recognition is one type of a classification task, where the input is an image, and the output is a numeric code identifying the bounding box around the object in the image.

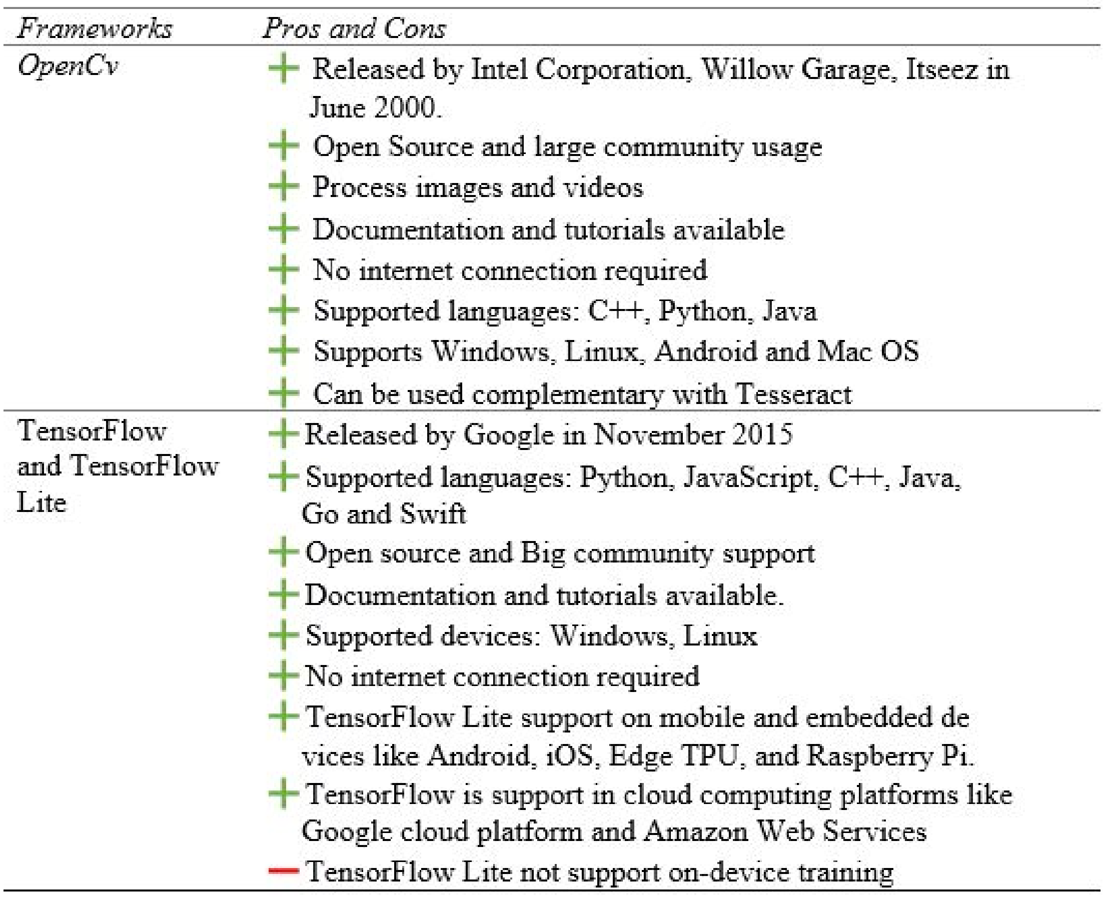
Object Detection and Object Recognition

Object recognition is the term used to describe a combination of related computer vision tasks that have the main goal of identifying objects in digital pictures. The process involves more than one task. It includes object detection, which as the name implies, is used to detect if there is an object in the picture. Then, it performs object localisation, which consists of locating one or more objects in an image and drawing a bounding box around them. Finally the recognition, also called classification, predicts the class of one or more objects in an image. It may be helpful to think of object discovery (instead of detection) and object comprehension (as a substitute for recognition) (Grevelink, 2017). For this project, those techniques will be applied to detect and classify which sort of tag it is being worked with.

Technologies

The application will be composed of an API that will be responsible for dealing with the HTTP requests sent from the front end. This API will contain all the object detection logic and also the character recognition processing before sending a response back to the client. The decision making process was based on comparisons between the available OCR libraries and Machine learning libraries. After extensive trials, the project solution will include a combination of the following technologies (see diagram in Appendix A).

Machine learning Libraries for Object Detection:



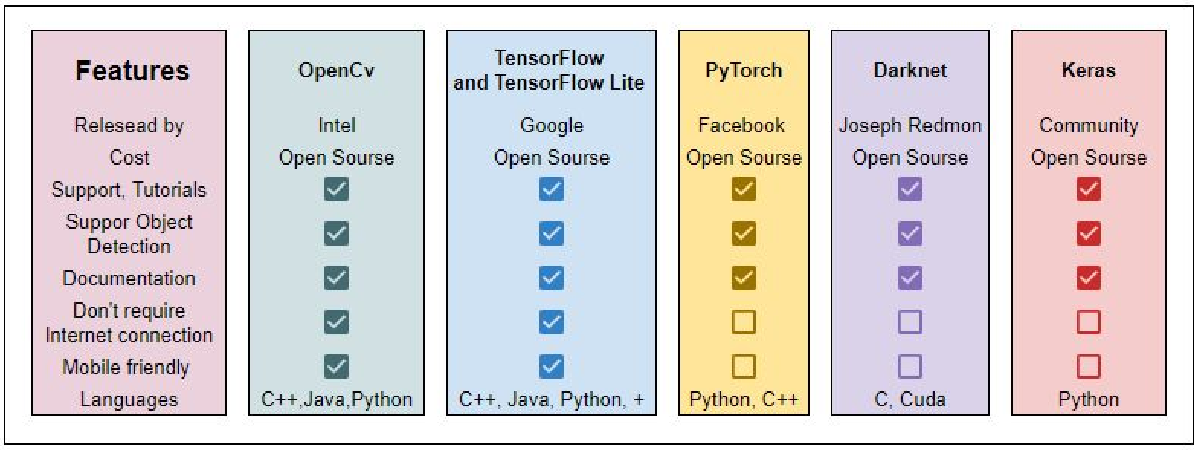
Frameworks	*Pros and Cons*
OpenCv	+ Released by Intel Corporation, Willow Garage, Itseez in June 2000. + Open Source and large community usage + Process images and videos + Documentation and tutorials available + No internet connection required + Supported languages: C++, Python, Java + Supports Windows, Linux, Android and Mac OS + Can be used complementary with Tesseract
TensorFlow and TensorFlow Lite	+ Released by Google in November 2015 + Supported languages: Python, JavaScript, C++, Java, Go and Swift + Open source and Big community support + Documentation and tutorials available. + Supported devices: Windows, Linux + No internet connection required + TensorFlow Lite support on mobile and embedded de vices like Android, iOS, Edge TPU, and Raspberry Pi. + TensorFlow is support in cloud computing platforms like Google cloud platform and Amazon Web Services − TensorFlow Lite not support on-device training

PyTorch	+ Released by Facebook + Documentation and Tutorials available + Open source + PyTorch is supported on Amazon Web Services, google cloud platform and Microsoft Azure. + Supported languages: Python 2, Python 3 and C++ − Supported devices: Desktop only
Darknet	− Released by Joseph Redmon + Can Run on CPU and GPU + Specialized in object detection + Open source + Good documentation and tutorials + Yolo was created in darknet − Supported languages: C and Cuda − Supported devices: Desktop Only
Keras	− Released by François Chollet Developer(s) various + Supported languages: Python + Supported devices: Desktop Only + Open source + Simple, easy to use and Runs seamlessly on CPU and GPU + Keras can use TensorFlow, Theano and, Microsoft Cognitive toolkit (CNTK) as backend. + Supports both convolutional networks and recurrent networks, as well as combinations of the two.

Table 2 - Libraries pros and cons

There are many options of machine learning libraries available, and the above table was made considering what would be a positive or negative point directly related to the requirements of this project. Through this table, it is easy to notice that OpenCV has no negative points and it would be the best fit to move forward with, but after further discussions, the team decided it was necessary a deeper comparison with only the essential characteristics to the development of this project.

Technologies comparison



Features	OpenCv	TensorFlow and TensorFlow Lite	PyTorch	Darknet	Keras
Relesead by	Intel	Google	Facebook	Joseph Redmon	Community
Cost	Open Sourse	Open Sourse	Open Sourse	Open Sourse	Open Sourse
Support, Tutorials	✓	✓	✓	✓	✓
Suppor Object Detection	✓	✓	✓	✓	✓
Documentation	✓	✓	✓	✓	✓
Don't require Internet connection	✓	✓	☐	☐	☐
Mobile friendly	✓	✓	☐	☐	☐
Languages	C++,Java,Python	C++, Java, Python, +	Python, C++	C, Cuda	Python

Image 2 - Technologies comparison chart

Conclusion

OpenCv and Tensorflow appear to be the two mobile friendly technologies that would be a perfect fit for this project. However, due to technical limitations, such as GPU power, alternative measures had to be taken on how to train the model. It was when Colab became a valuable tool to complete this project. The IDE played an important role in the training process and without it, the accomplished work would not reach the same level as it did. The downside was that the development has become far too dependent on the tool. These factors have guided the decision of which technologies to use, reason being that Darknet was the main library.

Darknet along with YOLOv3 make a fast and precise object detection combo and have enough material available on how to use them with Colab. Even though Tensorflow was not the technology we moved further with, research shows it is possible to convert the darknet model into a Tensorflow model. In his repository, the developer of Darknet, AlexeyAB, provides instructions on how to migrate a Darknet model to be used with Tensorflow (AlexeyAB, 2019). In conclusion, converting the Darknet model would be the preferable approach to the problem, but as a result of the time window of this project, the efforts had to be aimed exclusively at training the model using Darknet and leave the Tensorflow approach for future development.

Colab

Google Colaboratory, or Colab for short, is a free Jupiter based notebook environment created by Google which provides free GPU and TPU processing online in the browser. It allows the combination of executable code and rich text in a single document, as well as images, HTML, LaTeX and more. In their own words, “Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education” and “providing free access to computing resources including GPUs”. (Google Colab FAQ)

Provided these features, Colab became an important tool in the model training. In order to train the YOLOv3 model in the custom dataset, computing resources like GPU are required since the process is very costly in terms of processing. Since none of the team members on this project had a powerful enough machine, all the training was accomplished using Colab's virtual machines.

Darknet

Darknet is a deep learning framework to train neural networks. It is open source and written in C/CUDA and serves as the basis for training YOLOv3 models. To rephrase it, it sets the architecture of the network. The framework has YOLOv3 as one of its features, a real-time object detection system and it can process images at 40-90 FPS. Darknet displays information as it loads the config file and weights. Then, it classifies the image and sends back all the classes contained in that image within their bounding boxes (Juanola, 2019). The main advantage of using Darknet is that it is written in C and CUDA and it takes advantage of GPU power, making it 500 times faster than other researched frameworks. And that is also the reason why it is being used in this project, to make the training and detection more efficient.

YOLOv3

YOLOv3 stands for “you only look once” version three and it is a real time object detection algorithm. As the name suggests the algorithm only looks at the frame once and applies a single neural network layer (Juanola, 2019). According to YOLOv3 developers, this network then divides the image into regions and predicts bounding boxes and probabilities for each region. which are weighted by the predicted probabilities.(YOLOv3) The main feature that differs YOLOv3 from other detection systems is that it uses only one layer in the neural network and makes predictions based on the context. In addition, it also makes predictions with a single network evaluation, different from systems such as R-CNN which require thousands of evaluations for a single image. This can make it more than 1000x faster than R-CNN and 100x faster than Fast R-CNN.

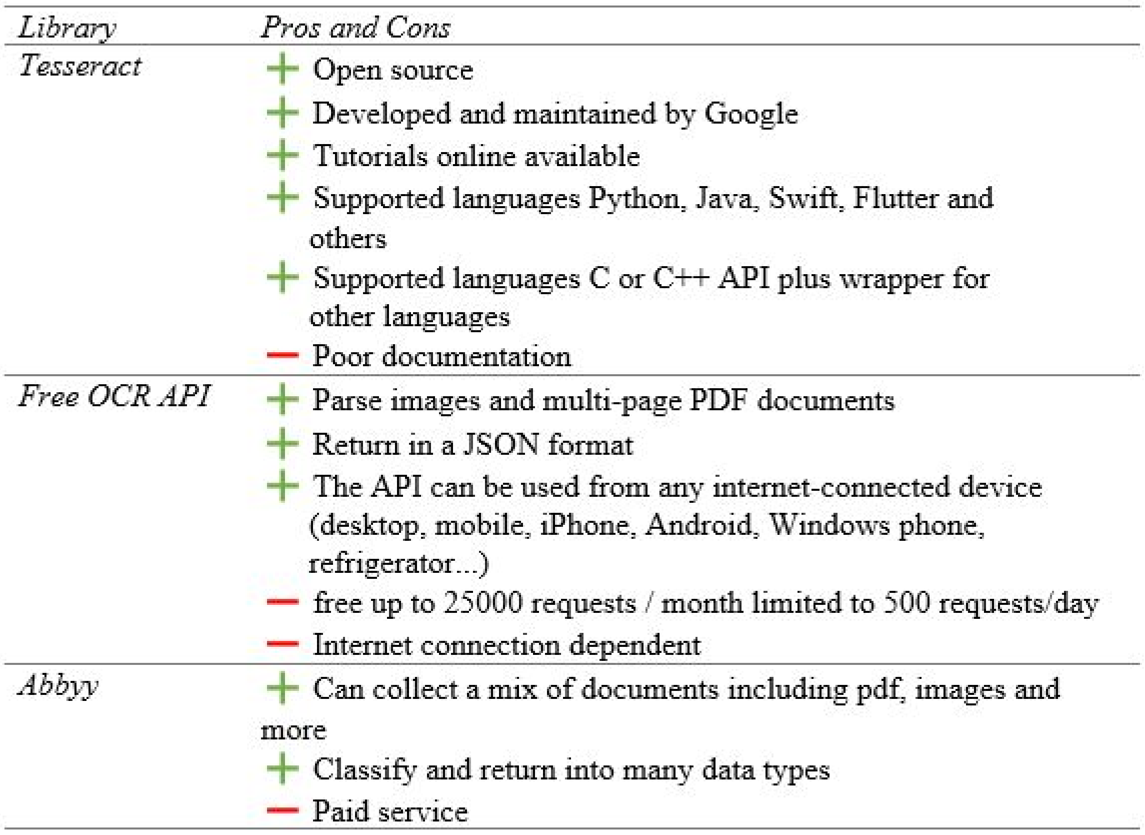
Due to the spatial constraints of the algorithm, it might not be the best solution when dealing with small objects, like a flock of birds. However, that limitation should not impact the results when applied to the scales’ tags dataset, because of the low complexity of the objects that need to be detected (YOLOv3). This was the selling point when YOLOv3 was chosen for object detection and recognition, the fact that it is fast and accurate when looking for not so complex objects. This feature fits exactly the need of the project.

OCR - Optical Character Recognition

Optical Character Recognition is a machine learning technology used to identify text characters and convert them into machine-encoded text. The technique is based on two major parts: text detection and text recognition. The Text Detection is the stage in which parts of the image that contain text characters are identified. The result can be achieved by two different approaches:

The Region-Based detectors first recognize those parts of the image containing text characters and then pass those to a classifier. This is a two-step process that (1) finds the bounding box and (2) finds the class of it. On the other hand, in the Single Shot approach, the detectors find the bounding box and the class at the same time. The result of the second approach is faster but can lose in accuracy when dealing with small objects, which is not the case here. Since the tags on the scales are quite similar in terms of the objects they contain, the precision on detections should be very high.

OCR Libraries



Library	*Pros and Cons*
Tesseract	+ Open source + Developed and maintained by Google + Tutorials online available + Supported languages Python, Java, Swift, Flutter and others + Supported languages C or C++ API plus wrapper for other languages − Poor documentation
Free OCR API	+ Parse images and multi-page PDF documents + Return in a JSON format + The API can be used from any internet-connected device (desktop, mobile, iPhone, Android, Windows phone, refrigerator...) − free up to 25000 requests / month limited to 500 requests/day − Internet connection dependent
Abbyy	+ Can collect a mix of documents including pdf, images and more + Classify and return into many data types − Paid service

Table 3 - OCR Libraries pros and cons

Conclusion

Tesseract has the most relevant characteristics for the development of this project. The tool is sponsored by Google and it has a big community support. Although, the most important aspect is the OCR stand alone functionality: the reason being the decision to move forward with Tesseract (see diagram in Appendix B)

Tesseract

Tesseract is an optical character recognition engine developed by Hewlett-Packard between 1985 and 1994 and released as open source in 2005. for various operating systems. It is a free software that works on Windows, Linux and macOS, and it is available under the Apache License. Since 2016, the project has been sponsored by Google (Tesseract Documentation).

In the current project, the Tesseract engine will be applied to extract the numeric characters from the objects which were recognized by the object detection. The reason for using Tesseract is that the algorithm is still considered the most accurate among the community, and also because it has been regularly improved, which opens the possibility for Application improvement.

cuDNN

Region-Based Convolutional Neural Networks, or R-CNNs, are a family of techniques for addressing object localisation and recognition tasks, designed for model performance. (Brownlee, 2019) The NVIDIA CUDA Deep Neural Network library is a GPU-accelerated library of primitives for deep neural networks. The framework provides highly tuned implementations for standard routines such as forward and backward convolution, pooling, normalization, and activation layers. cuDNN is advertised as a tool that allows for easy multithreading and interoperability with CUDA streams. This allows for the developer to explicitly control the library setup, and ensure that a particular GPU device is always used in a particular host thread, for example. cuDNN was used in the training with the purpose of reducing the model training time.

Creating Datasets

As with any machine learning based project, a dataset has to be provided to the system. The amount of images may vary for each project, but if the number of images is not enough, the use of Image Augmentation is recommended. Features that are part of the Keras framework can perform augmentations which can increase the number of samples by changing the size, light, contrast and other attributes of the pre-existing data set.

Considering that Accuscience's scale calibration service handles a few different types of equipment - each one identified by a specific label format - the creation and manipulation of the personalized dataset will be clarified in the next section.

Training the Model

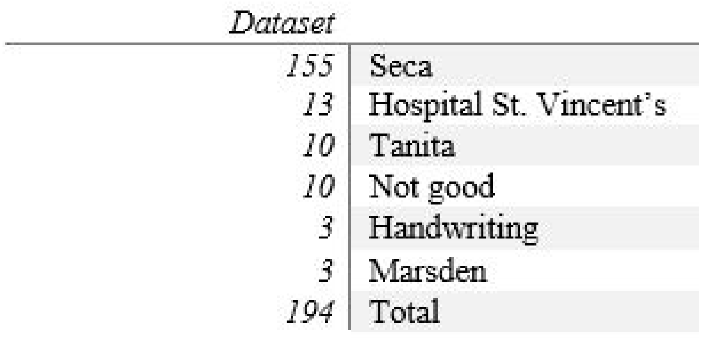
An essential part of the present project is to run the training of the custom model that was specifically created for this purpose. This model must then identify the fields in the labels, such as the serial number, asset number and the scale model number. The steps below need to be carried out.

Dataset preparation

The dataset is the core of any machine learning model. Still, Goodfellow, Bengio and Courville affirm that the central obstacle in machine learning is that it needs to provide good performance on new, previously unseen inputs - not just those on which the model was trained. “The ability to perform well on previously unobserved inputs is called generalisation.” (Goodfellow, Bengio and Courville, 2016.). There are a range of factors that can negatively influence the learning process of the machine, such as the size of the dataset, multiclass classification and unbalanced classes. The hits and the setbacks of the data preparation for the model will be discussed in the following sections.

Dataset collection

This stage of the project was of collaboration between the team, together with the Accuscience engineers. They contributed by taking as many pictures of labels as possible and by the middle of March 2020, the dataset contained almost 200 pictures in total.



Dataset	
155	Seca
13	Hospital St. Vincent's
10	Tanita
10	Not good
3	Handwriting
3	Marsden
194	Total

Table 4 - Final dataset distribution

Adversities

There were three major challenges in the dataset collection stage, one cascading into another. Starting with how highly the production of the pictures was dependent on the engineers schedule and disponibility. It was relying 100% on the engineers' to provide the pictures of the labels, which were taken in between calibrations, as an extra task.

The second factor was a consequence of the first, since Accuscience's calibrations schedule were not as frequent as expected. This created a very tight time frame to work with, even though the collection of data started in the very early stage of the process. As soon as the need for a custom dataset was identified, Accuscience staff was requested to start taking as many pictures as possible of the labels on the scales they were working on.

The third and most unexpected complication was the Covid-19 crises. Since the lockdown started, the activities at Accuscience had to be suspended and with that, the collection of the dataset was also stalled. This had a huge impact on development and imposed a limitation on the size of the dataset.

Dataset Size

When discussing the size of a dataset, Goodfellow, Bengio and Courville state that a "rough rule of thumb is that a supervised deep learning algorithm will generally achieve acceptable performance with around 5,000 labeled examples per category". (Goodfellow, Bengio and Courville, 2016) They also discuss two of the main challenges related to the dataset size: Underfitting and Overfitting.

Underfitting happens when the model does not have enough data to find discriminative features. The model then is not capable of distinguishing the new data (Goodfellow, Bengio and Courville, 2016). The easiest solution for this case would be collecting more distinct raw data, but in situations where this option is not possible, the augmentation of the dataset can be an alternative to overcome this constraint.

Overfitting, on the other hand, occurs when the model memorizes repetitive features in the training dataset and it becomes really good at identifying data that has the same features as the trained ones, but very bad at identifying new input data (Goodfellow, Bengio and Courville, 2016). One way to make sure the prediction results are accurate, is to separate part of the dataset to be used for testing the accuracy of the predictions. Having a set of images that are not part of the pictures known to the model will help prevent false positives when running tests.

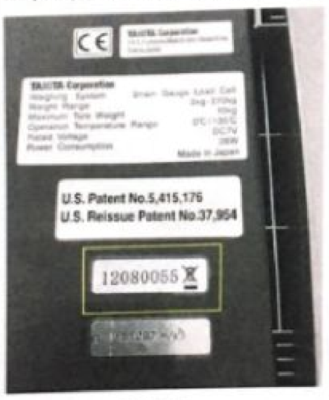
Low quality data

The creators of YOLO start their article stating that "Poor data quality is enemy number one to the widespread, profitable use of machine learning." (Redmon, Divvala, Girshick and Farhadi, 2016) In order to guarantee the quality of the dataset, the team went through the 194 images and categorized them. Finished with this analysis, some of the images were deemed as not legible or somehow unfit for the training, summing the total of 10 unused samples.

Identifying Classes

In the context of machine learning, classes are like the categories of objects present in the dataset. When the dataset is being prepared, the specific objects in the picture are selected and a class is assigned to that object. These classes are used as a reference for the classifications of the object when running the predictions. In this stage, the team had to examine the images in the dataset in detail to identify the classes we will be using to guide the training of the model. Following the documentation provided by accuscience, the findings derived from this analysis and examples can be found below. These are documents handed by Accuscience to help identify important features and information associated with them.

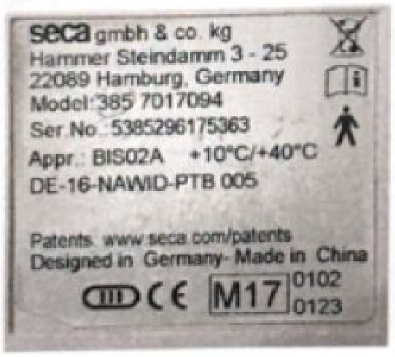
Sample equipment label – Manufacturer TANITA



Serial number Highlighted

12= year 2012, **08**= Month 8 ie August, **0055**= unique ID.

Sample equipment label – Manufacturer SECA



Serial Number **5385296175363**

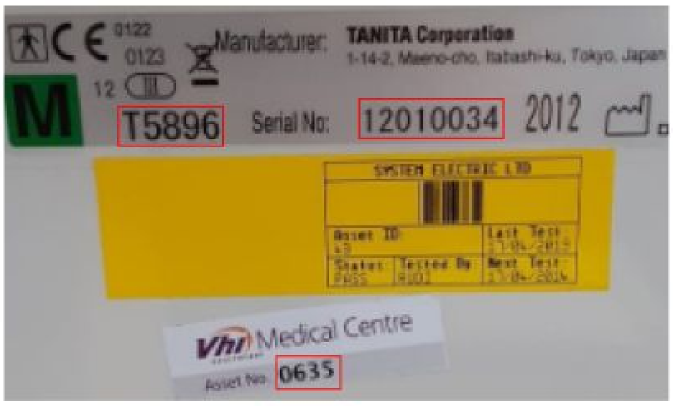
5=Factory, **385**=model, **26917**= day 269 of year 2017, **5363**=unique id

Model No 385

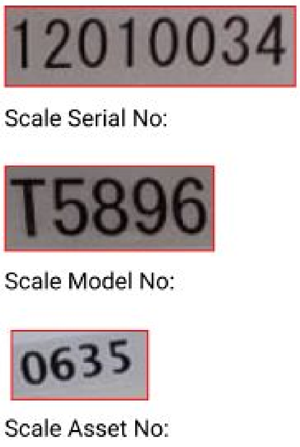
Image 3 - Accuscience tags explanation

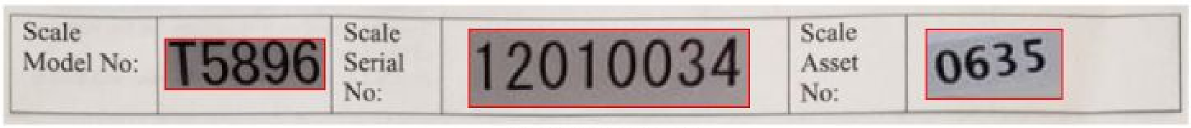
Accuscience Documentation X Dataset: A Comparison

Tanita Samples



Tanita label





Scale Model No:	T5896	Scale Serial No:	12010034	Scale Asset No:	0635

Accuscience Form

Image 4 - Fields identification on labels

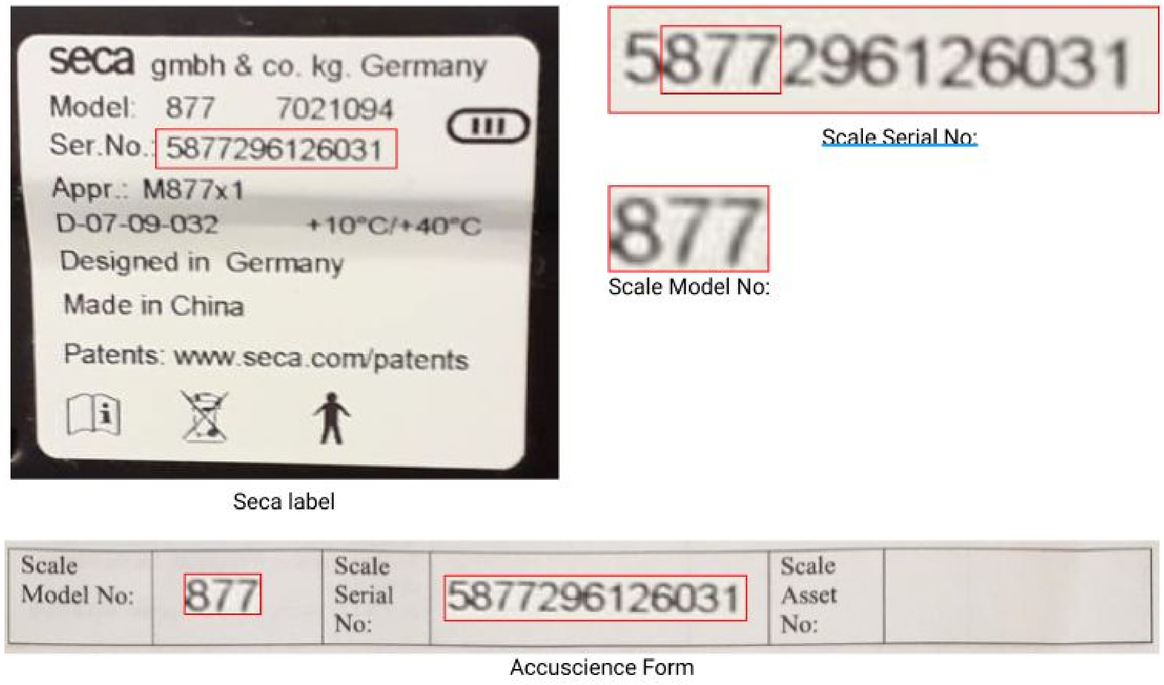
By looking at Tanita's images (4) it was possible to locate the serial number right after the *Serial No* tag, even though the document given by Accuscience shows an 8 digits field with no labels as the serial number and no reference of the model number. Reaching out to Accuscience engineers the researchers were informed that in some of the Tanita's models, the serial number can be presented as the ones contained in the dataset.

From the exemple above it is possible to visualize the asset number in the VHI medical center label attached right below Tanita's vendor information. The asset number tag can also be found in some samples of Seca dataset images and it represents an internal identification of this facility.

Adversities

The team contacted Tanita company to double check if the number we considered as a model was indeed the Tanita model type, but until the end of this project, there was no reply from them. However, Accuscience's engineers confirmed that the characters shown above are the sequence they considered the model.

Seca Samples


5877296126031
Scale Serial No:
877
Scale Model No:
Seca label
Accuscience Form

seca gmbh & co. kg. Germany
Model: 877 7021094
Ser.No.: 5877296126031
Appr.: M877x1
D-07-09-032 +10°C/+40°C
Designed in Germany
Made in China
Patents: www.seca.com/patents

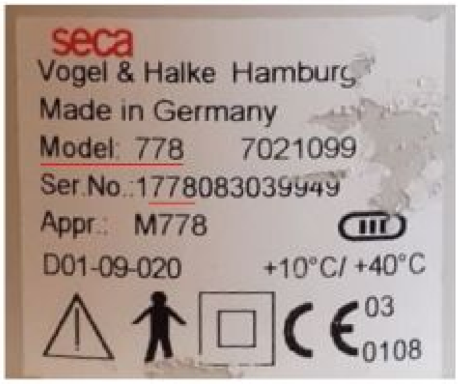
Scale Model No:	877	Scale Serial No:	5877296126031	Scale Asset No:	

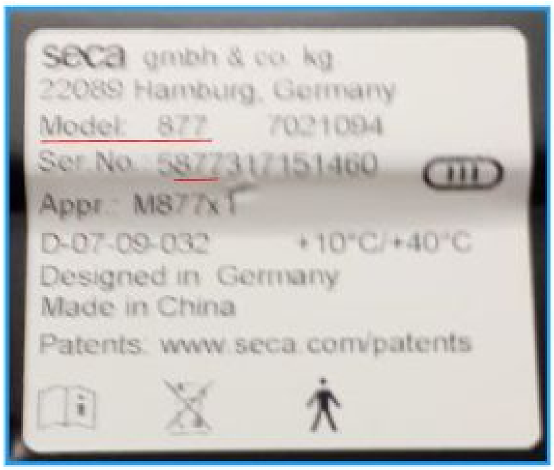
Image 5 - Fields identification on Seca's labels

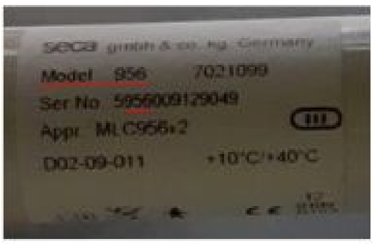
In the seca tag the characters that follow the field “Ser.No.:” are the most relevant information in the tag. From this sequence of numbers the Seca model number can be extracted.

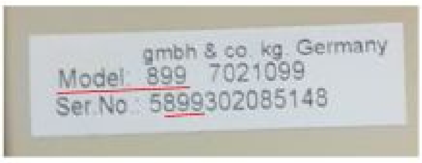
Adversities

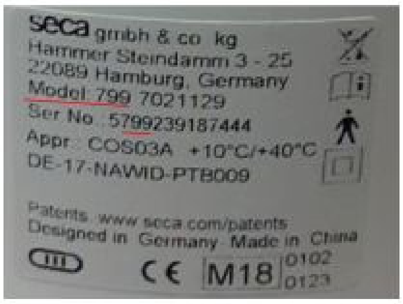
In the provided documentation, Accuscience engineers detailed how the serial number is composed and how they identify which model they are calibrating. To confirm this information, Seca vendor was contacted by email but with no replies. Nevertheless, this pattern can be observed by checking the Seca dataset. The first digit represents the vendor, the second, third and fourth digits represent the model of the scale. The remaining digits are the date it was made and the unique id as seen below.


seca
Vogel & Halke Hamburg
Made in Germany
Model: 778 7021099
Ser.No.:17780830399949
Appr.: M778
D01-09-020 +10°C/ +40°C
CE 03 0108


seca gmbh & co. kg
22089 Hamburg, Germany
Model: 877 7021094
Ser.No.: 5877317151460
Appr.: M877x1
D-07-09-032 +10°C/+40°C
Designed in Germany
Made in China
Patents: www.seca.com/patents


seca gmbh & co. kg Germany
Model 956 7021099
Ser No 5956009129049
Appr MLC956+2
D02-09-011 +10°C/+40°C


gmbh & co. kg. Germany
Model: 899 7021099
Ser.No.: 5899302085148


seca gmbh & co. kg
Hammer Steindamm 3 - 25
22089 Hamburg, Germany
Model: 799 7021129
Ser.No.: 5799239187444
Appr.: COS03A +10°C/+40°C
DE-17-NAWID-PTB009
Patents: www.seca.com/patents
Designed in Germany · Made in China
CE M18 0102 0123

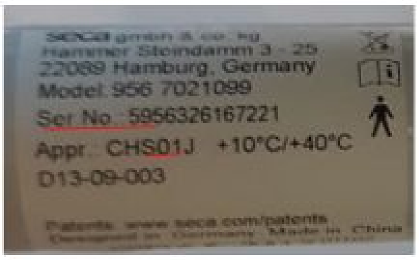
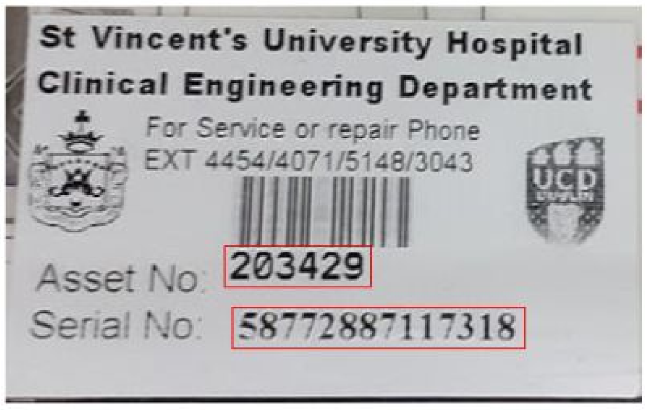

seca gmbh & co. kg
Hammer Steindamm 3 - 25
22089 Hamburg, Germany
Model 956 7021099
Ser No 5956326167221
Appr CHS01J +10°C/+40°C
D13-09-003
Patents: www.seca.com/patents
Designed in Germany Made in China

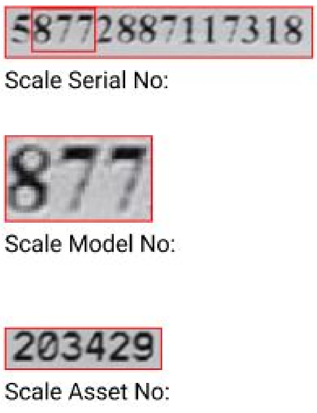
Image 6 - Model identification though serial number on Seca’s labels

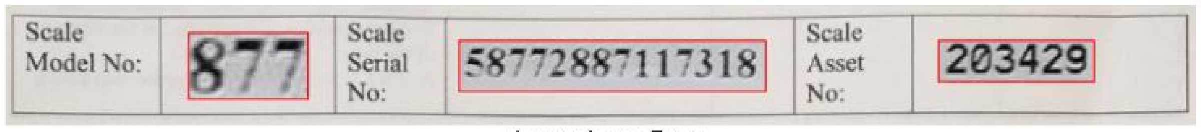
Hospital St. Vincent images



St Vincent's University Hospital
Clinical Engineering Department
For Service or repair Phone
EXT 4454/4071/5148/3043
Asset No: 203429
Serial No: 58772887117318

St. Vicent's University Hosptal label





Scale Model No:	877	Scale Serial No:	58772887117318	Scale Asset No:	203429

Accuscience Form

Image 7 - Fields identification on St. Vincent's Hospital labels

St. Vincent's Hospital uses only Seca scales. Even though they use a custom tag on their scales, they are still identifiable by the serial number and the model number, which correspond to the pattern used for Seca scales.

Classification Outcomes

From this analysis four different classes were extracted : seca_serial, tanita_serial, tanita_model and asset_number. The process also helps with the decision to, momentarily, not work with the tags from Marsden, since they are off the market. The handwritten tags were also left for a future stage of this project, since it would add a degree of complexity too high for the number of image samples available.

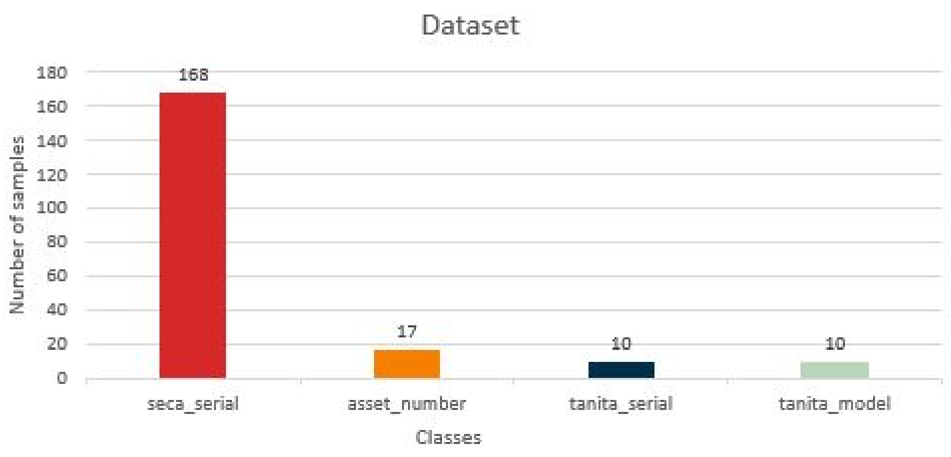


Image 8 - Number of samples per class

Multiclass Classification

When classifying a dataset, having multiple classes can present a different challenge then a simple binary classification. The unequal distribution can make a lot of conventional machine learning systems less effective, especially in predicting minority class examples (Guo and Viktor, 2004). A classification task with more than two classes makes the assumption that each object sample is assigned to one and only one label: e.g. an object defined by a bounding box can only belong to one class.

Unbalanced classes

Guo, and Viktor, described unbalanced classes as: “The class imbalance problem corresponds to domains for which one class is represented by a large number of examples while the other is represented by only a few. ...When learning from imbalanced data sets, machine learning algorithms tend to produce high predictive accuracy over the majority class, but poor predictive accuracy over the minority class" (Guo and Viktor, 2004).

The preferable solutions for this problem would be gathering more samples of the underrepresented classes, although when this solution is not possible, augmentation techniques can be applied. Another alternative is to under-sample or remove some samples of the over-represented classes to have more balanced classes.

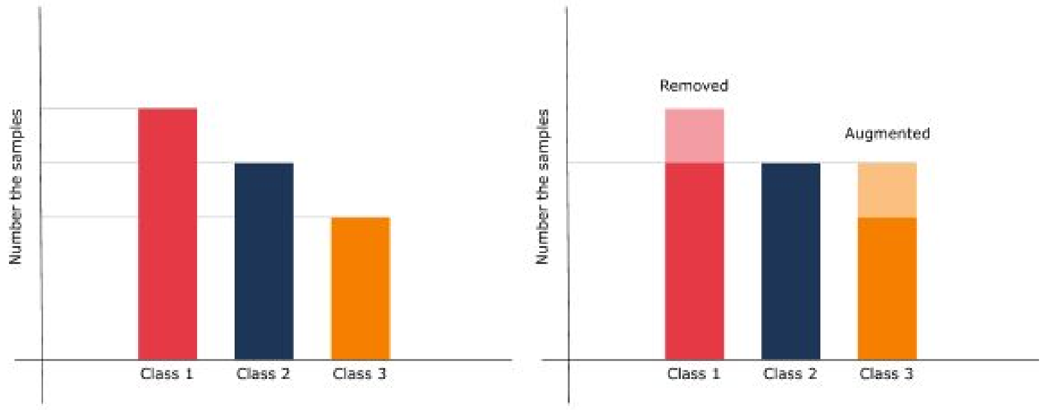


Image 9 - In the example above, a generic dataset has three unbalanced classes, which are manipulated to contain roughly the same number of samples utilised

Dataset Augmentation

In order to have a more balanced dataset, the ImageDataGenerator class from Keras framework was utilised. The snippets below represent the Kera ImageDataGenerator being applied to the Tanita dataset and the same procedure was applied to tags which contained the asset_number class. This code was written using Python language. The first snippet is listing the imports of libraries required to perform the dataset augmentation.

```
#Read from the directory and use the augmentation
import keras
from numpy import expand_dims
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.preprocessing.image import ImageDataGenerator
from matplotlib import pyplot
from keras.preprocessing.image import save_img
```

Now the modifications that will be done to the images are being passed as parameters to the ImageDataGenerator. For this case, the Generator will perform a rotation range, width shift, height shift, zoom, and the fill mode is set to constant.

```
# create image data generator
datagen = ImageDataGenerator(
#Int. Degree range for random rotations
rotation_range=15,
#int:  integer  number  of  pixels  from  interval  (-width_shift_range,
+width_shift_range)
width_shift_range=[0.1, 1.0],
#int:  integer  number  of  pixels  from  interval  (-height_shift_range,
+height_shift_range)
height_shift_range=[0.1, 1.0],
#Tuple  or  list  of  two  floats.  Range  for  picking  a  brightness  shift
value from.
brightness_range= [0.2, 1.0],
# Float or [lower, upper]. Range for random zoom. If a float, [lower,
upper] = [1-zoom_range, 1+zoom_range]
zoom_range=[0.3, 0.8],
#One of {"constant", "nearest", "reflect" or "wrap"}. Default is
#'nearest'. Points outside the boundaries of the input are filled
#according to the given mode: 'constant': kkkkkkkk|abcd|kkkkkkkk,
#'nearest': aaaaaaaa|abcd|dddddddd 'reflect': abcddcba|abcd|dcbaabcd,
#'wrap': abcdabcd|abcd|abcdabcd
fill_mode="constant"
)
```

Next, the location of the dataset samples, the specifications of the batch size, and the configurations of how the new images have to be saved and where have also to be passed as attributes.

```
# load the image
dir = "/mydrive/MAD_Code/Colab Notebooks/images/"
# flow_from_directory gets a folder and itinerate. It also saving into
the given directory
it = datagen.flow_from_directory(
    dir,
    #default target_size=(256,256)
    target_size=(1280,626),
    #number of the entries, it make one batch we have 10 samples in the
    #tanita database so 10 is the batch size
    batch_size=10,
    #shuffle : false because we don't want the risk of repeat the image
    #when running the batch
    shuffle=False,
    #save to the given directory
    save_to_dir=dir,
    # specify the format of the new images and the prefix
    save_format='jpg',
    save_prefix='Tanita_'
    )
```

Finally, the range that represents how many times the generator will run is defined.

```
# range : quantity of times each batch will run
for i in range(12):
  # generate batch of images
  batch = it.next()
  # convert to unsigned integers for viewing
  image = batch[0].astype('uint8')
```

The Results

The aftermath from running this code can be seen in the examples below. The images had attributes like rotation range, width shift, height shift and zoom manipulated to replicate the dataset. The final result for Tanita is a dataset that now has 120 samples. The asset_number class has also been augmented and it contains now 136 images. The example below shows the augmentation of one batch.

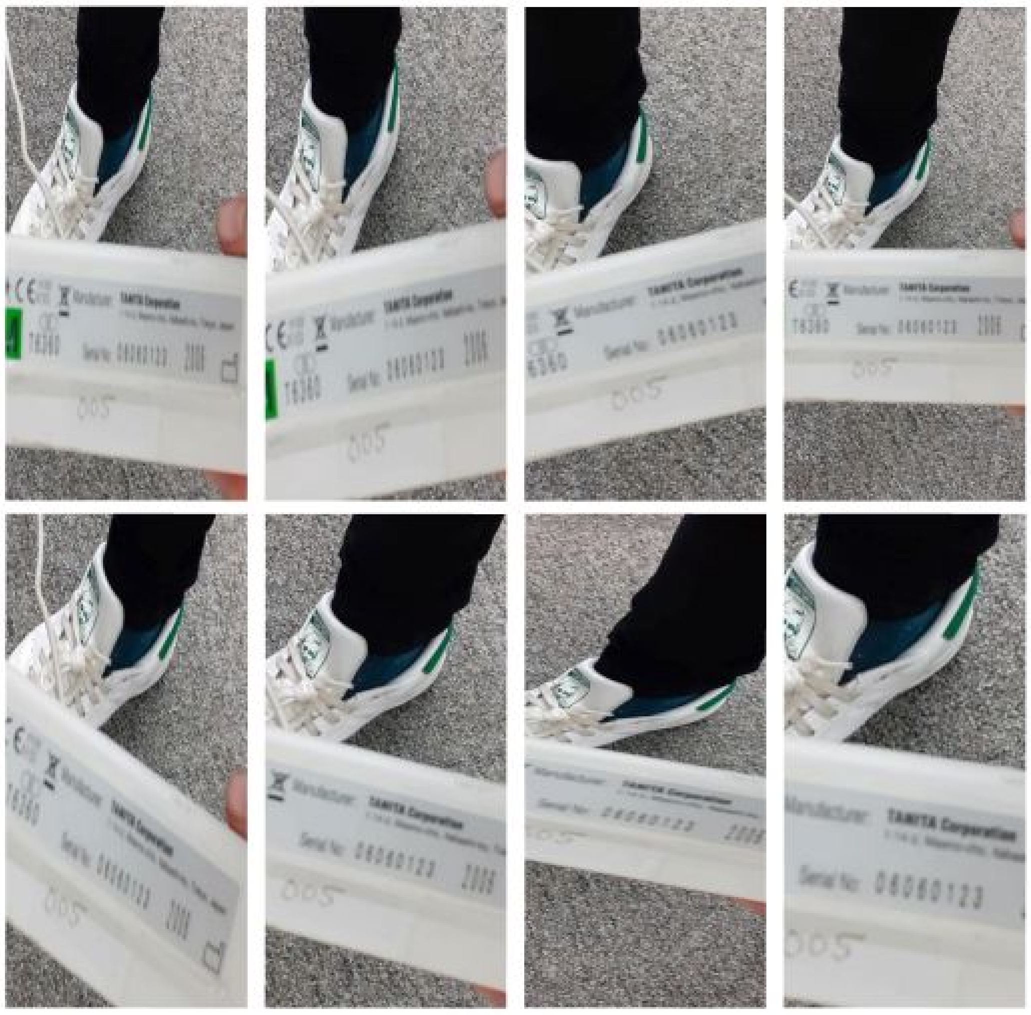


Image 10 - Results of the augmentation process of a Tanita tag, multiplying one picture by eight

Loss

After the augmentation process, not all images are fit to be used to train the model. Some of them came out with cropped information or not legible, and so had to be discarded.

Class	*Batch x Range*	*Sub-Total*	*Loss*	*Total*
seca_serial	168	168	0	168
asset_number	17*8	136	23	113
tanita_serial	10*12	120	20	100
tanita_model	10*12	120	20	100
Total		544	63	481

Table 5 - The total amount for each object class

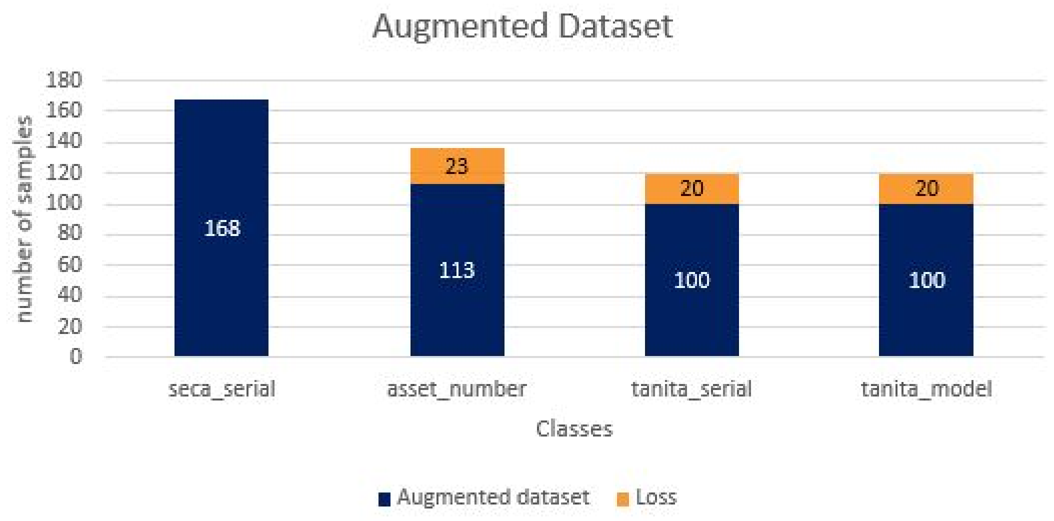


Image 11 - Augmented Dataset

Conclusion

This process made it possible to obtain a more balanced dataset and better predictions on the Darknet model. As seen in Image x the difference in numbers between the four classes is now considerably smaller than before going through augmentation. This action helped dealing with the issue of unbalanced classes, previously mentioned.

Training and Testing Datasets

This is the part of the process when the dataset had to be separated between the training set and the testing set. The training set corresponds to the data that is going to be used to train the model while the test set is used after the model is trained, to run prediction and check its accuracy. When extremely few labeled training examples are available, Bayesian neural networks outperform dropout on the Alternative Splicing Dataset, where fewer than 5,000 examples (Goodfellow, Benjio and Courville, 2016).

It's advisable to also have a validation set, used to tune the model, although given the size of the current dataset, it was decided that as much data as possible would be used for the training. On the current dataset, the proportion of 80% for training and 20% for the test was applied to all classes.

Training

The produced dataset is now used to improve the model's ability to predict which type of label was captured by the camera. The process begins by sending some input data to the model, which will attempt to predict the outcome for that data. The model predictions are then compared with the output which should have been produced, and with each iteration - or training step - the model improves the ability to predict an expected outcome.

To train a YOLOv3 model using Darknet framework, the following configuration files are required:

Dataset Labeling

It is now the moment to start with the data annotation of each and every image of the final training and test dataset. The fist YOLOv3 model that was trained on the dataset had a total of 481 manually labelled images after the augmentation While researching different options that were compatible with YOLOv3, the program LabelImg proved to be the best and more straightforward interface to use for this matter. The creator of LabelImg describes it as "...a graphical image annotation tool. It is written in Python and uses Qt for its graphical interface. ... Besides, it also supports YOLOv3 format" (Lin, 2020). The process consists of setting the preset classes as the classes defined in the classification stage, and, after selecting a rectangle around the desired object, one must select each of the preset classes that object belongs to. In the images below, one can see how this process works.

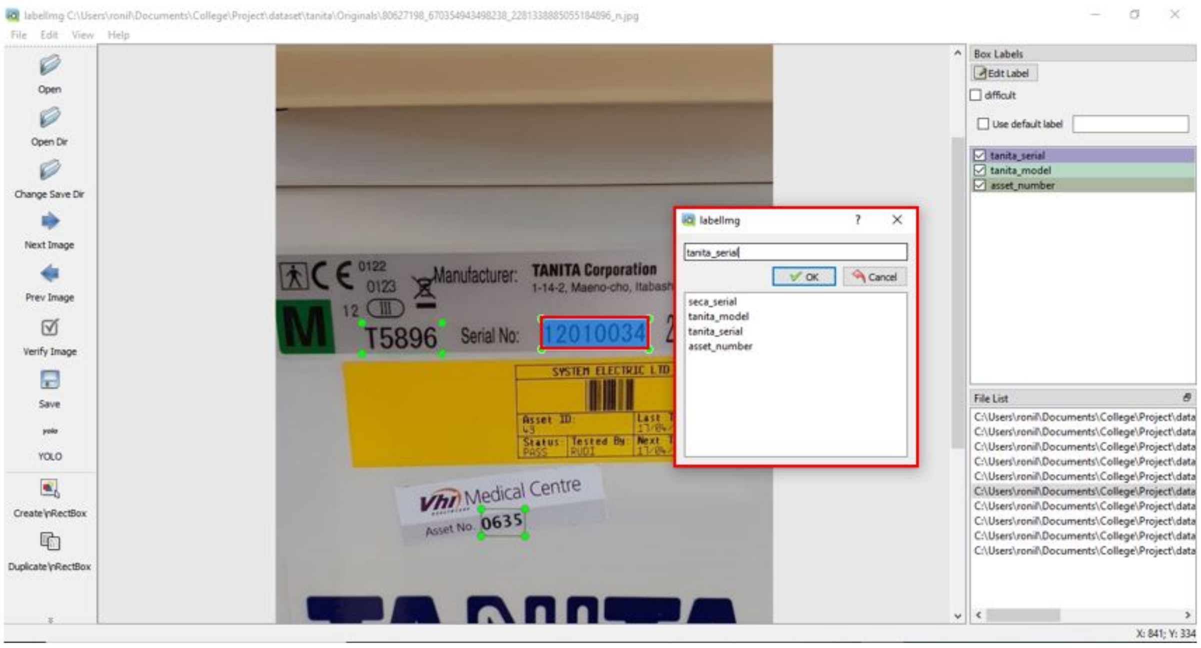


Image 12 - Labeling a Tanita tag that has a serial and a model number as its classes

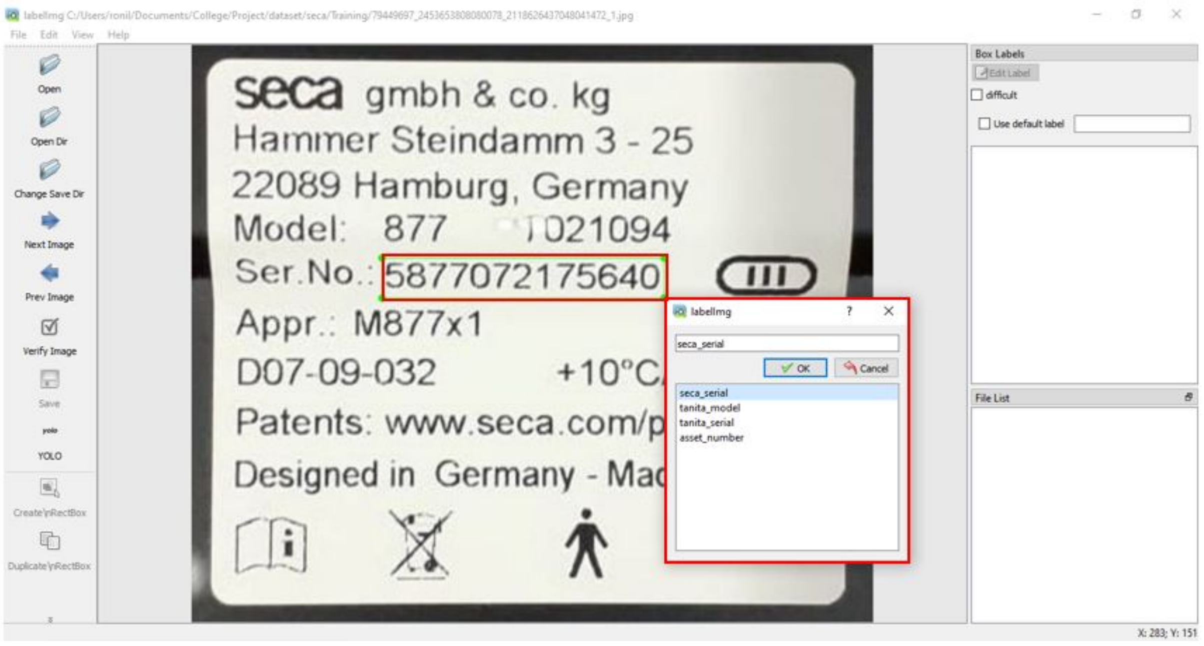
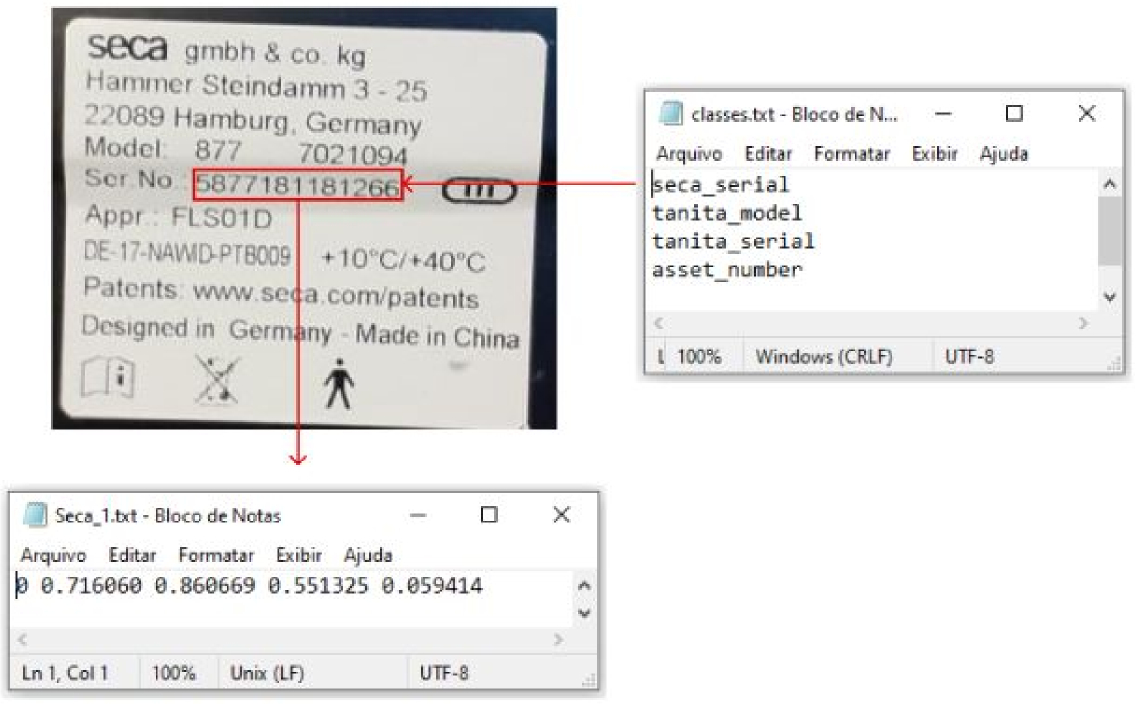


Image 13 - Labeling a Seca tag where the serial number has all the details about the scale

Results

The results after each image is labeled is a .txt file created for each one of the pictures. These files will have the same name as its corresponding image, and contain the YOLOv3 label coordinates of the objects in the format *class_index box_x_ratio box_y_ratio box_width_ratio box_height_ratio* (see in the picture below). With that done, the program will also generate a classes.txt with the classes used for the labeling.


seca gmbh & co. kg
Hammer Steindamm 3 - 25
22089 Hamburg, Germany
Model: 877 7021094
Ser.No.: 5877181181266
Appr.: FLS01D
DE-17-NAWID-PTB009 +10°C/+40°C
Patents: www.seca.com/patents
Designed in Germany - Made in China
classes.txt - Bloco de N...
Arquivo Editar Formatar Exibir Ajuda
100% Windows (CRLF) UTF-8
Seca_1.txt - Bloco de Notas
Arquivo Editar Formatar Exibir Ajuda
Ln 1, Col 1 100% Unix (LF) UTF-8

```
seca_serial
tanita_model
tanita_serial
asset_number
```

```
0 0.716060 0.860669 0.551325 0.059414
```

Image 14 - The image above shows the labeled Seca tag, the .txt files with the the index and coordinates for locating the object in the correspondent image, and on the top right, the classes.txt files with all possible classes

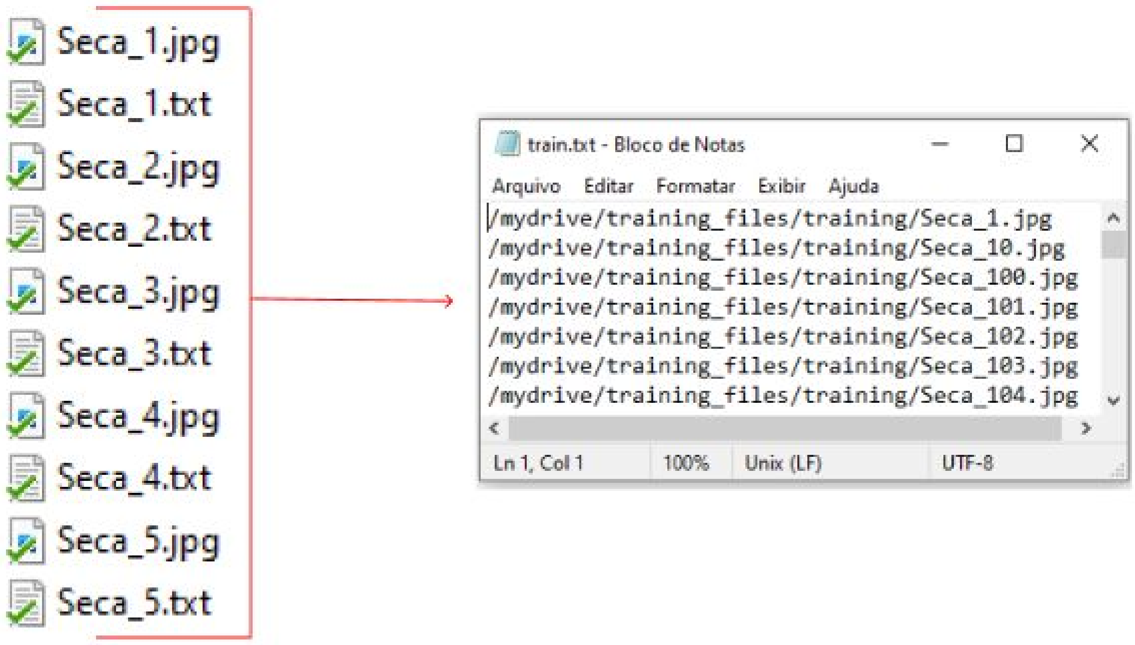
Configuration Files

The official Darknet github repository provides the step by step of the process to set the configuration to train a YOLOv3 model. The training of a personalised YOLOv3 model requires the following configuration files. See the full graph with the relationship between the configuration files in the Appendix C.

File: train.txt and test.txt

The train.txt is responsible to direct the model to the every image that the model will be exposed to. (AlexeyAB/darknet, n.d.) To do so the train.txt file will contain all the relative paths of each image in the training dataset.

The test.txt has a similar role, to show the path to the test images, although these sample images will not be used in the training but later, together with the “-map”, a Darknet flag to test the model’s accuracy. To create those paths manually for each image would be very laborious, so this task was delegated to a little Java program to generate the paths. The snippet of this code can be found in the Appendix D. The outcome from the Java program is the train.txt and test.txt with a personalised path to each image file., as shown below:



```
/mydrive/training_files/training/Seca_1.jpg
/mydrive/training_files/training/Seca_10.jpg
/mydrive/training_files/training/Seca_100.jpg
/mydrive/training_files/training/Seca_101.jpg
/mydrive/training_files/training/Seca_102.jpg
/mydrive/training_files/training/Seca_103.jpg
/mydrive/training_files/training/Seca_104.jpg
```

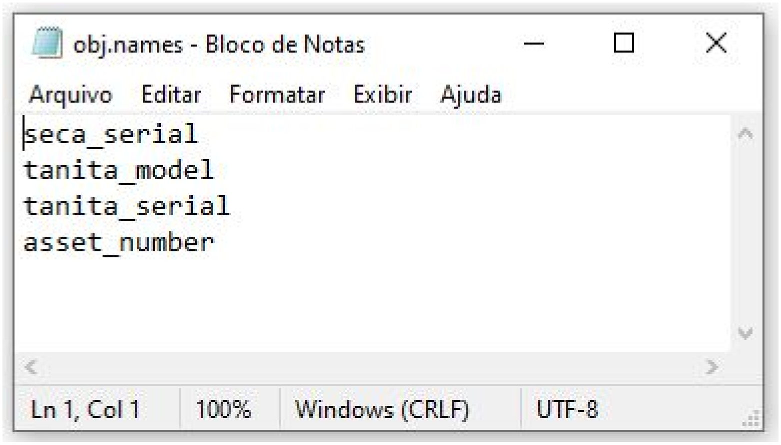
Image 15

Adversities

The learning curve necessary for the development of this project produced a few setbacks. One of them was the limitations in the usage of Colab, such as the resources "vary over time to accommodate fluctuations in demand, as well as to accommodate overall growth and other factors" (Colab FAQ). Also, the service can be rather volatile and it enforces a maximum lifetime limit and deletes the virtual machine when idle for some time. That means that any files or installations performed in the Colab's virtual machine are not kept. The action taken to overcome that issue and avoid redoing a substantial amount of work was to have an account on Google Drive and create a folder where all the configuration files would be saved and loaded to the Notebook.

File: object.names

The object.names file must have the classes' names in the exactly same order as they are listed in the classes.txt file. The order of the classes in the file is very important, because is how the model relates the YOLOv3 localisation with its corresponding classes.



obj.names - Bloco de Notas
Arquivo Editar Formatar Exibir Ajuda

```
seca_serial
tanita_model
tanita_serial
asset_number
```

Ln 1, Col 1
100%
Windows (CRLF)
UTF-8

Image 16 - obj.names

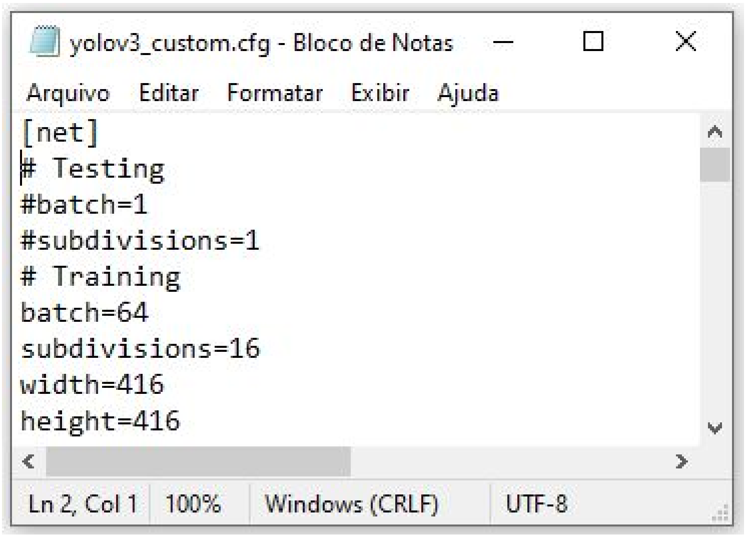
File: yolov3_custom.cfg

The YOLOv3 configuration file must be edited with the corresponding settings to run the custom dataset. The first step is to download the file from the YOLOv3 repository, and change its name from *yolov3.cfg* to *yolov3_custom.cfg*.

```
# download cfg and change its name
```

```
!cp cfg/yolov3.cfg /mydrive/yolov3/yolov3_custom.cfg
```

With the file open, comment the *batch=1* and *subdivisions=1* in lines three and four, and uncomment the *batch=64* and *subdivisions=16* in the lines six and seven. Then, set *batch=64* and *subdivisions=16* if they are not already set to those values.

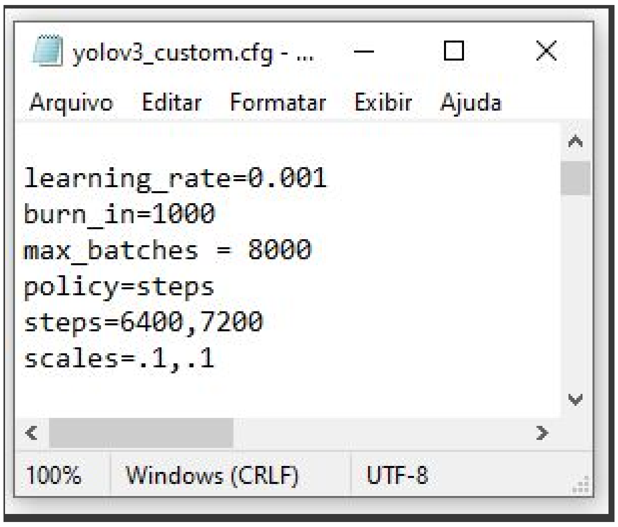

yolov3_custom.cfg - Bloco de Notas
Arquivo Editar Formatar Exibir Ajuda

```
[net]
# Testing
#batch=1
#subdivisions=1
# Training
batch=64
subdivisions=16
width=416
height=416
```

Ln 2, Col 1 100% Windows (CRLF) UTF-8

Image 17 - yolov3_custom.cfg

Next, scrolling down in the same file, change the max_batches to be 2000x the number of classes in the dataset. In this case, the model has four classes, so max_batches will be 8000. The steps also need to be adapted for the custom dataset amount of classes. The values for the steps should be 80% of the max batches for the first parameter and 90% for the second separated by a coma "," (6400,7200).


yolov3_custom.cfg - ...
Arquivo Editar Formatar Exibir Ajuda

```
learning_rate=0.001
burn_in=1000
max_batches = 8000
policy=steps
steps=6400,7200
scales=.1,.1
```

100% Windows (CRLF) UTF-8

Image 18 - yolov3_custom.cfg: configuring batches

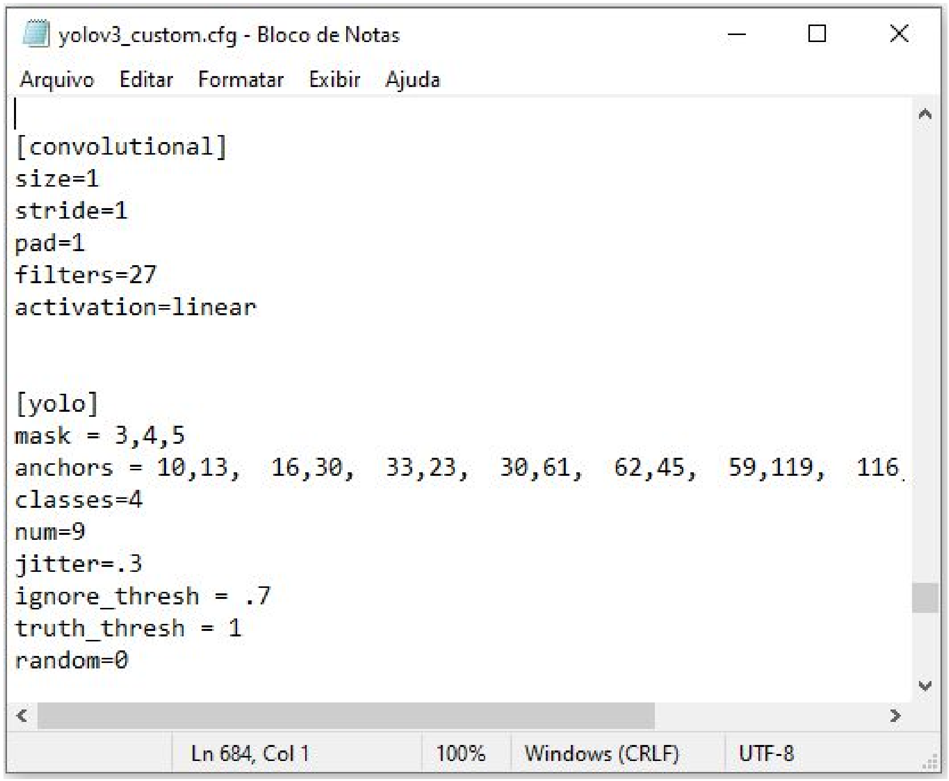
Final step to configure the yolov3_custom.cfg file is:

- Search for all YOLOv3 the references in the file (3 references in total).
    - change the number of classes to the amount of classes in the dataset (4).
- Above every YOLOv3 reference, change the convolutional filters. They should follow the formula *filters = (n_of_classes + 5)x3*.

yolov3_custom.cfg overview

- **batch=64:** is the number of samples (images) which will be processed in one batch.

- **subdivisions = 16:** is the number of mini_batches in one batch, size mini_batch = batch/subdivisions. GPU processes mini_batch samples at once, and the weights will be updated for batch samples.
- **max_batches = 8000:** the training will be processed for this number of iterations (batches)
- **steps = (6400,7200):** setting steps to 80% and 90% of max_batches
- **classes = 4:** changing classes=80 to the number of object classes corresponding the dataset
- **filters = 27:** editing the number of filters should follow the formula (classes + 5)x3 i.e. (4 + 5)x3 for this case. (AlexeyAB, 2020)

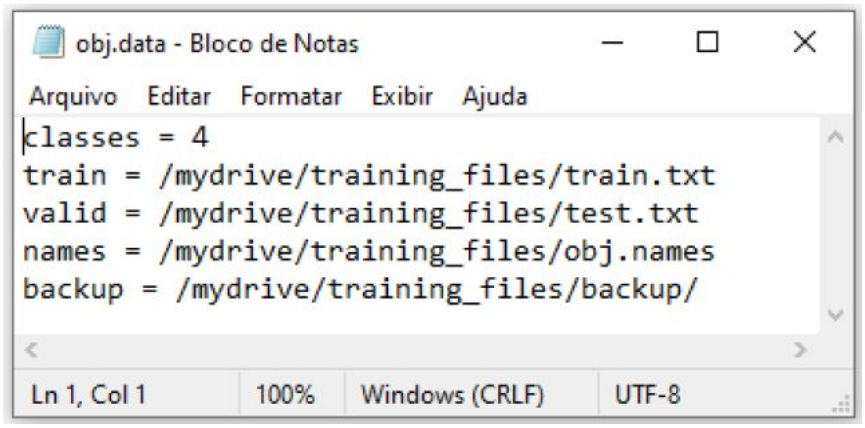

yolov3_custom.cfg - Bloco de Notas
Arquivo Editar Formatar Exibir Ajuda
Ln 684, Col 1
100%
Windows (CRLF)
UTF-8

```
[convolutional]
size=1
stride=1
pad=1
filters=27
activation=linear


[yolo]
mask = 3,4,5
anchors = 10,13,  16,30,  33,23,  30,61,  62,45,  59,119,  116
classes=4
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=0
```

Image 19 - yolov3_custom.cfg: classes and filters

File: obj.data

The obj.data file must contain the number of classes to be trained and the relative path from where to find all the configuration files. The folder backup is optional and was added to save the results of the training (weights).


obj.data - Bloco de Notas
Arquivo Editar Formatar Exibir Ajuda
Ln 1, Col 1
100%
Windows (CRLF)
UTF-8

```
classes = 4
train = /mydrive/training_files/train.txt
valid = /mydrive/training_files/test.txt
names = /mydrive/training_files/obj.names
backup = /mydrive/training_files/backup/
```

Image 20 - obj.data with the configuration related to the personalised dataset

Running Model Training

With all the configuration files created and edited to match the dataset characteristics, the environment needs to be set in Colab Notebooks and there are important configurations to do before starting the training of the model. The training will be performed in Colab because it offers free GPU processing on its virtual machines, making it possible to run the training without an actual computer with GPU enabled.

Mounting Google Drive

The snippet below will mount the user's Google Drive account into the Notebook and give Colab permissions to access its content within the project.

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

Symbolic Link

To facilitate the path to files inside Google Drive, a symbolic link was created to shorten its link.

```
#Symbolic link
!ln -s /content/gdrive/My\ Drive/MAD_Code/ /mydrive
!ls /mydrive
```

Helpers

The online community has provided the project with some good insights and also resources. The Python functions below were written by Ivan Goncharov and are used to display, upload and download images (Goncharov).

```
# define helper functions
def imShow(path):
  import cv2
  import matplotlib.pyplot as plt
  %matplotlib inline

  image = cv2.imread(path)
  height, width = image.shape[:2]
  resized_image = cv2.resize(image,(3*width, 3*height), interpolation =
cv2.INTER_CUBIC)

  fig = plt.gcf()
  fig.set_size_inches(18, 10)
  plt.axis("off")
  plt.imshow(cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB))
  plt.show()

# use this to upload files
def upload():
  from google.colab import files
  uploaded = files.upload()
  for name, data in uploaded.items():
    with open(name, 'wb') as f:
      f.write(data)
      print ('saved file', name)

# use this to download a file
def download(path):
  from google.colab import files
  files.download(path)
```

To make the model run faster, the use of GPU was enabled in the virtual machine's settings. The following code is also downloading the convolutional layer weights for the YOLOv3 network. Using the YOLOv3 weights reduces training time and increases accuracy.

```
# Run this code just once to download Darknet, update its
# configurations, compile and compile to drive directory
# Comment this code on the future runs.
!git clone https://github.com/AlexeyAB/darknet/

#change directory to update files to enable OPENCV, GPU and CUDNN
#%cd darknet
!sed -i 's/OPENCV=0/OPENCV=1/' Makefile
!sed -i 's/GPU=0/GPU=1/' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/' Makefile

#Compile Darknet
!make

#Copies the Darknet compiled version to Google drive
%cd ..
!cp -r ./darknet mydrive/MAD_Code/training_files/bin/darknet

# Uncomment after the first run, when you have a copy of compiled
Darkent in the Google Drive

#Makes a dir for darknet and move there
!mkdir darknet
%cd darknet

# Copy the Darkent compiled version to the VM local drive
!cp /mydrive/training_files/bin/darknet ./darknet

# Set execution permissions to Darknet
!chmod +x ./darknet
```

Here, the YOLOv3 weights for the convolutional network layers are downloaded to improve the time taken in the training and the accuracy of the predictions.

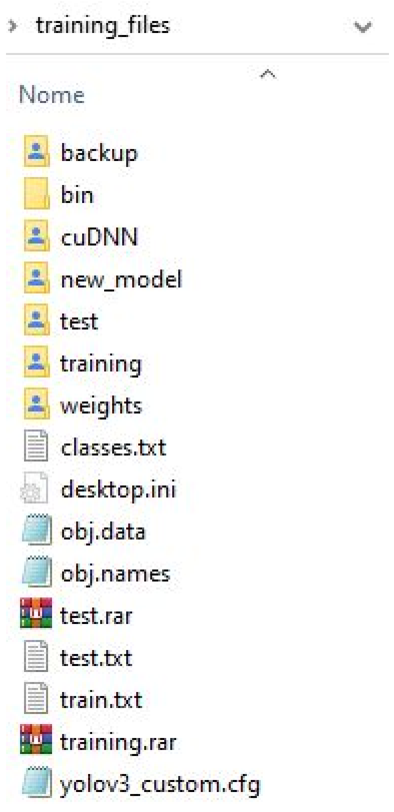
```
# Download the pretrained convolutional layer weights only once then
# Comment this code

# download pretrained convolutional layer weights
!wget http://pjreddie.com/media/files/darknet53.conv.74
# copie to drive
!cp -vr darknet53.conv.74 mydrive/MAD_Code/training_files/weights
```

Adversities

Preparing the environment for the training can be quite time consuming. After repeating this process a few times, the need for optimizing the download and compiling of the weights became clear. To avoid this rework, the weights were also moved to the Google Drive. This action reduced the time of loading the weights in ten times

Training Results

This is the final step towards training a custom model. It is the actual training itself. Darknet will run a custom training using the configuration files previously mentioned and the labelled images. On this stage the training_files folder structure looks like this:


training_files

Nome

backup
bin
cuDNN
new_model
test
training
weights
classes.txt
desktop.ini
obj.data
obj.names
test.rar
test.txt
train.txt
training.rar
yolov3_custom.cfg

Image 21 - training_files folder structure and contents

```
!./darknet  detector  train  "  <path  to  obj.data>"  "<path  to  custom config>" "<path to darknet53.conv.74>" -map -dont_show
```

The command above will start the training of the model. It runs the train command inside Darknet followed by the path to the obj.data path, the path to the custom_config file and the path to the weights for the training.

Adversities

Colab is set to go to sleep mode after 30-90 minutes of inactivity, which means they interrupt the runtime. Training the model takes long hours and this limitation would make it impossible to accomplish the training. The below snippet should be pasted into the browser's console. The function clicks in the window every ten minutes, preventing the page from going idle, and so, avoiding interruption on the training.

```
# This is a snipped from the AIGuy

function ClickConnect(){
console.log("Working");
document.querySelector("colab-toolbar-button#connect").click()
}
setInterval(ClickConnect,60000)
```

Even using the code above, the training process needs to be supervised and it can stall at any moment. If it happens, it is possible to restart the training from where the point it stopped by using the yolov3_custom_last.weights saved in the previously created backup folder.

```
# Start training at the point where the last runtime finished
!./darknet detector train <path to obj.data>" "<path to custom config>"
"yolov3_custom_last.weights" -map -dont_show
```

Model

In the course of this development, two models were trained. The first one had ten classes and the augmentation procedure was applied to all samples of the dataset, without considering the balance between the number of images in each class. The second model was reduced to only four classes and had the classes balanced by the augmentation algorithm. Another difference worth noting is that the training of the first model took 48 hours in total, while the second model took only 12 hours to finish the process.

Training results

The following line of code is what is needed to run a detection with Darknet.

```
# show predictions
!./darknet detector test " <path to obj.data>" "<path to custom
config>" "yolov3_custom_final.weights" "image.jpg" -dont_show
```

First model

The first model was trained during the research process because it was a dependency for other modules of the project to move forward and because the model could be improved later on. For this reason the team decided to go ahead even without the full understanding of the technologies to proceed with the training.

Because of the lack of previous knowledge, this model used the augmentation methodology on all the dataset, including Seca images, when it should not. After this process, a dataset with a total of more than a thousand images was ready to train the model. 10% of those images were used to test the model and the other 90% to train the model. Another issue with the first model is that no caution related to low quality images or unbalanced classes was taken. The only goal was to understand the mechanics of the training.

For the reasons described above this first model was considered the Test model. Unfortunately by the time of this first training the team held no knowledge of the generation of a chart with the loss and accuracy numbers for the model however after the training is finished the model prints the final accuracy and loss. For this model the accuracy was 68.2% and the loss 0.183.

Testing

After running the training, the results when checking prediction's precision are tested using a different set of images, not previously known by the algorithm. In image 22, the predictions ran quite well and returned meaningful results but on image 23, the process was not capable of finding the right location of the objects and its classes.

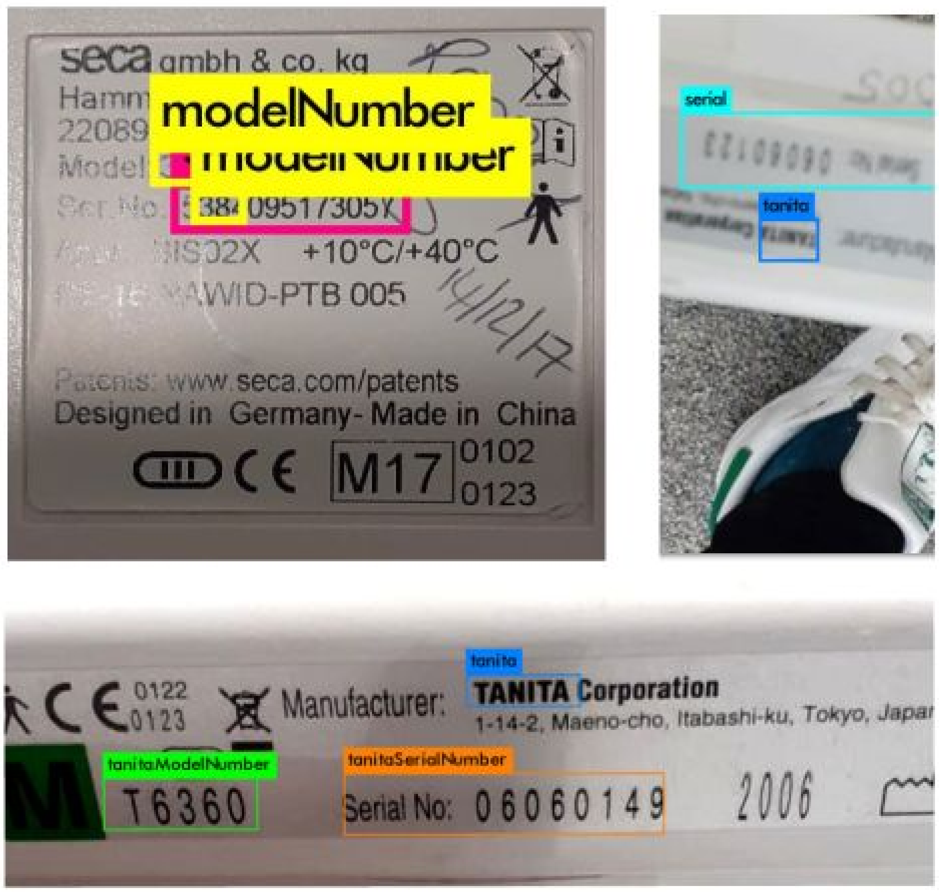


Image 22 - Predictions results showing the objects and its bounding boxes

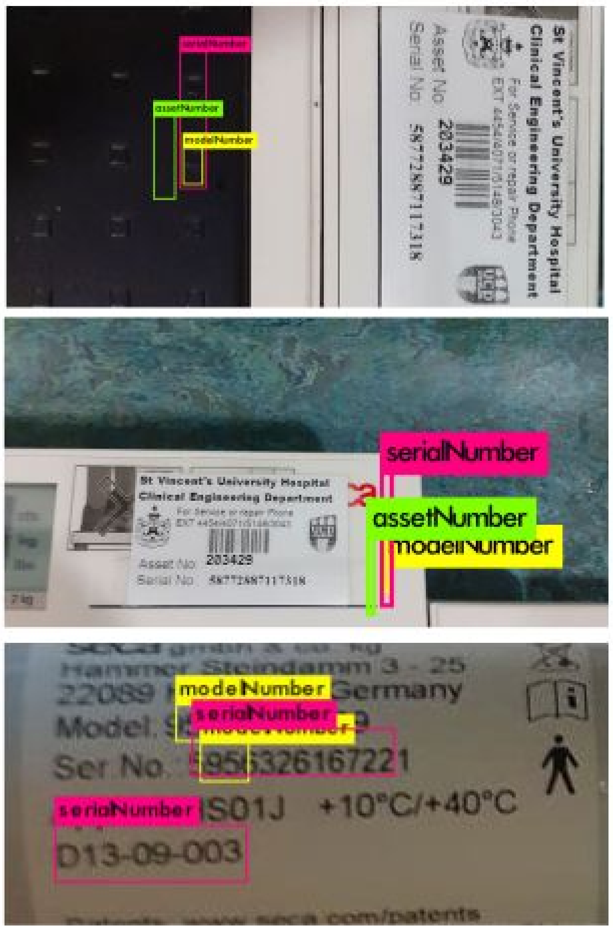


Image 23 - Bad predictions results showing the objects with rogue bounding boxes

The first model training took nearly 48 hours to be complete, and the result of 62% accuracy was higher than what was expected, considering all the processes that were skipped. Despite the accuracy not being too low, it is possible to notice that the first attempt to manipulate the dataset resulted in an overfitting model. Observing the image 23, the model is giving the prediction of a position where it believed the Seca serial number should be present. It appears that the model learned the position of the object, not what are the attributes of the object. The cause for the misclassification seems to be related to the position of the objects in the image. The asset_number class dataset had too many pictures with the asset_number on the same position, which caused an overfitting of the model.

Adversities

Because of Colab's volatility, the virtual machine was shut down during the training and the connection was lost. As mentioned earlier, a backup folder saves the weight results so the training can be resumed, from where it stopped.

Second model

The second model followed all the steps and procedures described in the above sections, and it gave a rather different outcome and the accuracy spiked to 92,6% and the loss also declined, but not as steep as the accuracy, which ended on 0.0823, close enough to the previous result. Here, one more time the connection was lost during the process, causing the training to be interrupted and then resumed. See below the training graph.

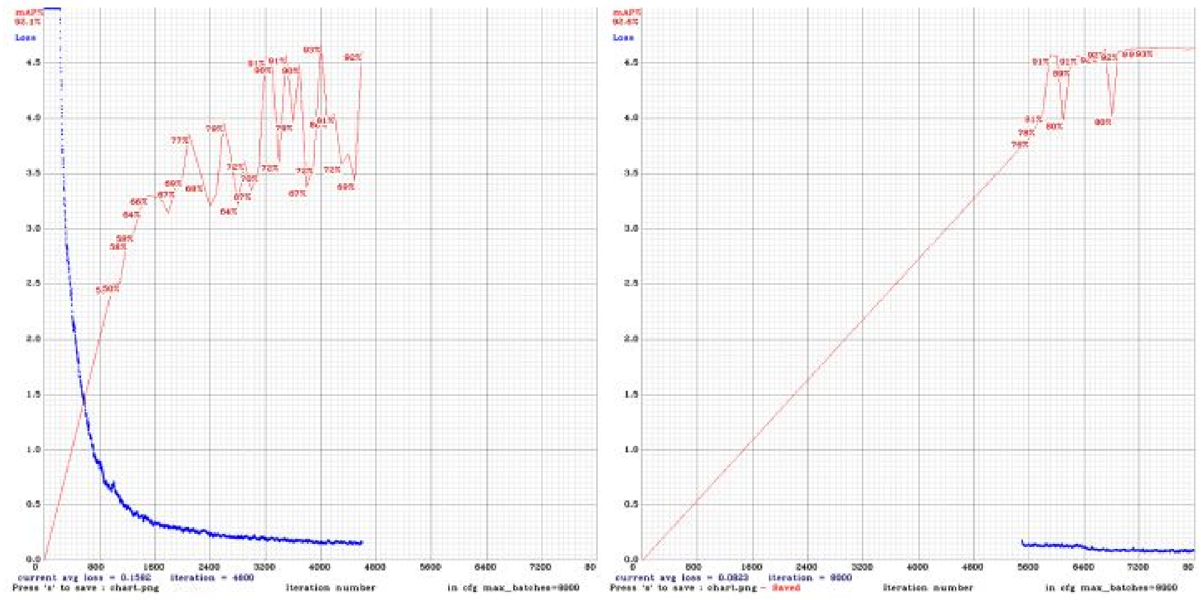


Image 24

Testing

On this turn, the test results are better than the ones from the first model. The model seems to be able to recognize the classes according to what it learned from the training.

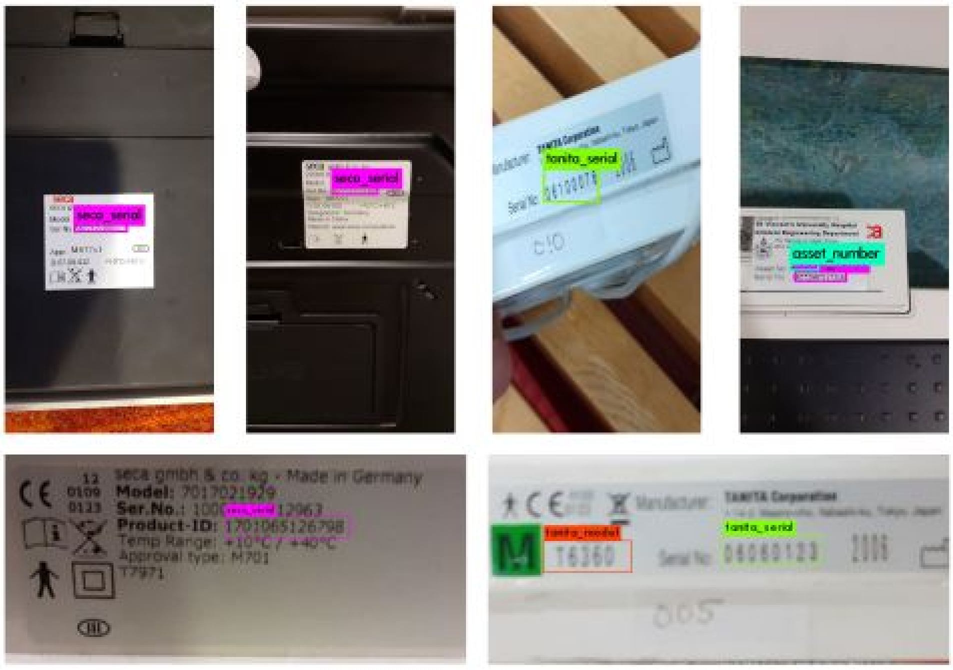


Image 25 - Label recognition

However, the model still struggles to identify the classes with the smallest amount of samples in the dataset.

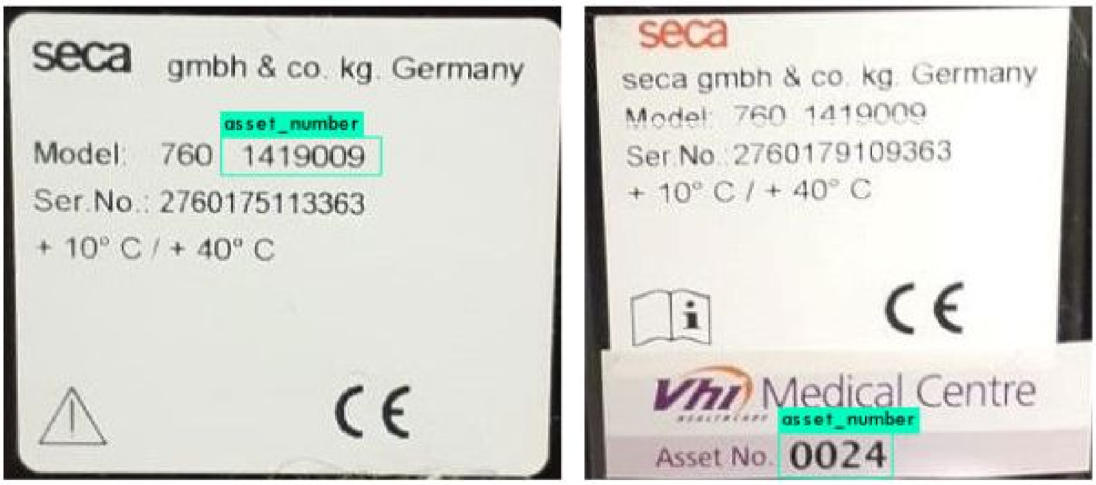


Image 26 - Wrong label identification

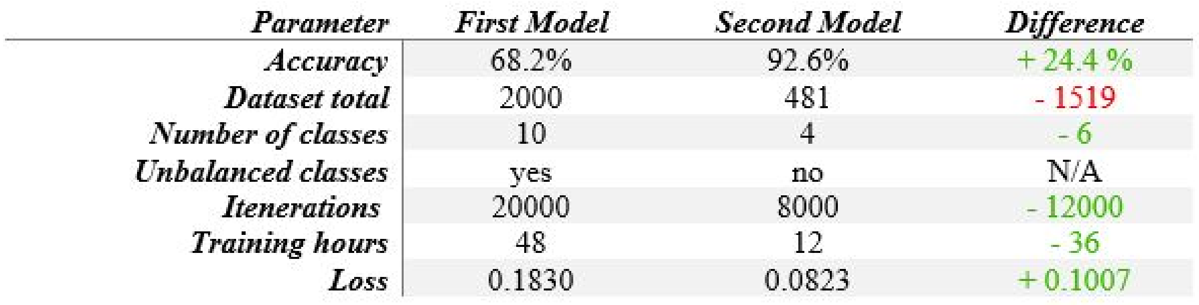
In the first example image 26, the prediction's accuracy is only 30% and the class assigned does not represent the real asset_number. Although the model understands that asset_number is a number, it is not accurate. It appears that if there were more samples of this specific class in the dataset, the model would be capable of distinguishing it.

In the second picture in image 26 the model identifies the asset_number correctly but in both cases the seca_serial class was not found. The cause might be that tags with both asset_number and seca_serial are shown together are in minority and the model did not learn to identify them together at this point.

Adversities

Once more during the training the connection was lost and the training had to resume from the backup weights. There were also a few missing labels on some images that might have been caused by the LabelImg tool. The problem happens if the classes.txt is not loaded automatically from the command line and the classes have to be manually inserted into the labeling tool. Doing it manually risks producing a file with typos or any little mistake and it messes with the order in which the classes appear, causing the model to classify the tags under a different class. To avoid this issue, after the all the dataset was labeled another member of the team had to verify if the tags correspond to the right class.

First Model vs Second Model



Parameter	*First Model*	*Second Model*	*Difference*
Accuracy	68.2%	92.6%	+ 24.4 %
Dataset total	2000	481	- 1519
Number of classes	10	4	- 6
Unbalanced classes	yes	no	N/A
Itenerations	20000	8000	- 12000
Training hours	48	12	- 36
Loss	0.1830	0.0823	+ 0.1007

Table 6 - Models comparison

Conclusion

The second model followed all the steps and measures toward the data and the final result had an improvement of 24% on the model's performance. One of the main problems of the first model is that it was overfitting which was overcome by reducing the number of classes and augmenting the under-represented classes to have a better balance among them. Reducing the number of samples helped to solve this specific issue, but the reduced number of samples clearly impacted negatively the second model, which now shows signs of being underfed.

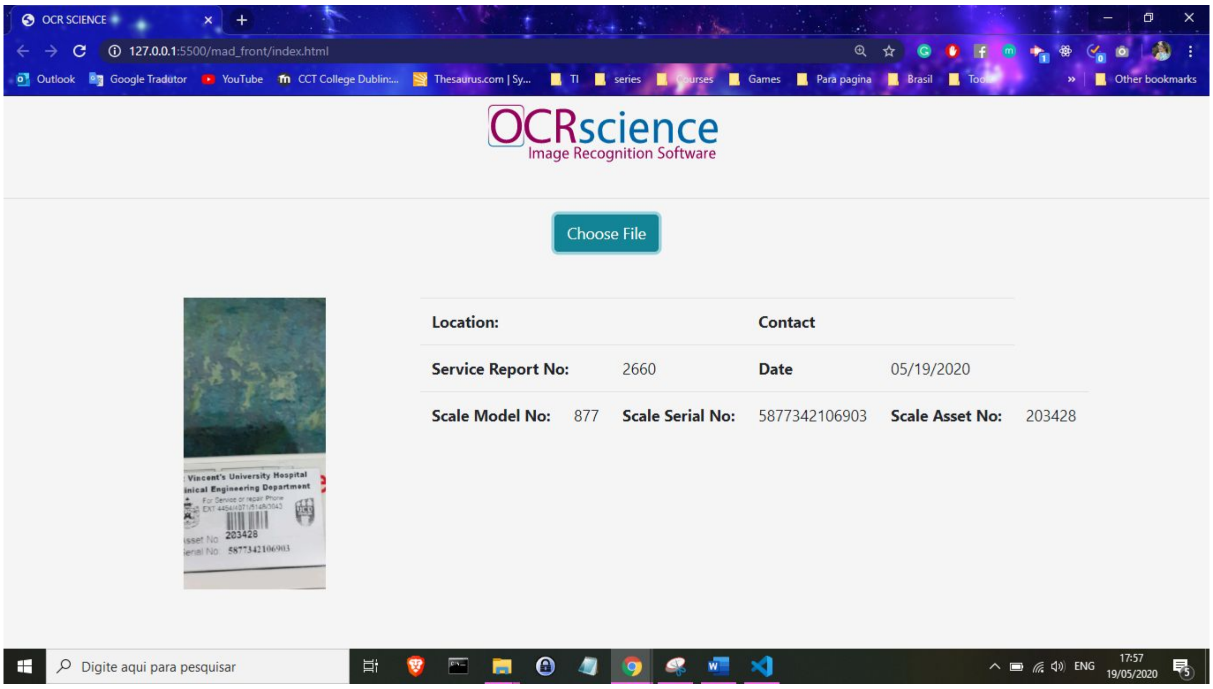
Nevertheless, proceeding with the first model without the clear understanding of the technologies was the right decision. It made it possible to understand the training mechanics and gain experience on what aspects to keep a closer eye on. Running the second model after doing the research was also extremely valuable, providing the knowledge to fix problems on the first model and to interpret the final results from a now more solid background. In conclusion, given the initial size of the database, the model was indeed expected to be underfed and the ideal solution would be to have a larger number of samples, even though under the current circumstances the second model is the best it can be giving satisfactory results.

The API

In order to have a universally accessible application to connect to the model, an API was developed. The API is written in Python and handles the tasks of the back end. The full circle of the application can be described as follows.

The first task of the API consists of handling the POST requests sent by the client-side. The method 'image_uploader' retrieves a base64 encoded text-file from the body of the request and sends it to the method 'toImage', which is responsible for turning the base64 back into a jpeg file and saving it inside the "temp" folder. If the conversion process is successfully completed the path to the jpeg file is sent as a parameter to the 'detect'. The 'detect' method starts by loading the image to extract some properties - like dimensions, channels and the number of pixels - and then creates the 4-dimensional blob necessary to obtain the predictions.

Once the detection process is finished and the output layers extracted, the bounding box of each detected object is used to create a cropped version from the original image. At getting an image for each detected object, the method precisely determines to Tesseract what has to be converted into characters and avoids a result containing a multiplicity of unnecessary data, which would have to be parsed in order to obtain only the relevant ones.


OCR SCIENCE
127.0.0.1:5500/mad_front/index.html
OCRscience
Image Recognition Software
Choose File
Digite aqui para pesquisar

Location:		**Contact**			
Service Report No:	2660	**Date**	05/19/2020		
Scale Model No:	877	**Scale Serial No:**	5877342106903	**Scale Asset No:**	203428

Image 27 - API demonstration

After Tesseract sends back the result of the character recognition for each cropped image, the method starts to compose the JSON that will be forwarded to the client. The representation of the detected object contains three fields: the class, which refers to the name of the detected object; the value, which is the result of Tesseract character recognition; and the confidence of the detection for that specific object. The final JSON will present from one to three objects depending on the label.

Conclusion

Although the internet is full of resources, it can be quite difficult to find free access academic material about some of the technologies which were used in the building of this project. The current lockdown situation worldwide as well imposed limitations on the second stage of the project (see Project Schedule). By interrupting the calibrations inspections at Accuscience, the collection of the custom dataset was also interrupted, causing the dataset to be smaller than it was first planned. This had a direct impact on the model training process.

Another setback overcome in the development of the application was Tesseract's restricted accuracy. While testing the algorithm, the struggle to read characters in pictures with poor resolution became clear. This will be a part of the development to be addressed in the future, to update the Tesseract version to the fifth and ensure that the system checks for the quality of the image before it sends it to object detection. For this prototype, a selection of images with good resolution had to be filtered for Tesseract's character recognition. Although the character recognition had issues with the images resolution, the low quality pictures could still be used for the training for the object detection.

Furthermore, delegate the character recognition process to the client-side should be considered. Steps like image cropping would also be removed from its executing cycle, leaving the API lighter and faster as its only responsibility would be the object detection process.

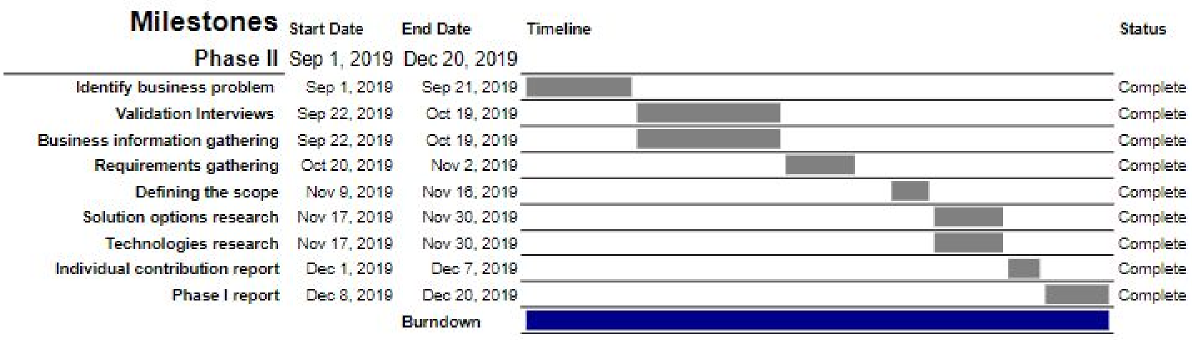
Following this approach, the client would receive a JSON containing the class, the confidence of the prediction and the bounding box of each detected object. The bounding box contained by the JSON could be used by a Tesseract wrapper like Tesseract.js to determine the specific image area to be converted into characters - and once again the cropped image process could be discarded.

The next stage for the full development of this project will be the implementation of continuous learning using AutoML pipeline to deal with the new image inputs while in production environment. A loop will be created to, after the system runs the prediction, save the image with its bounding boxes back to the dataset to retrain the model periodically (AlexeyAB, 2020). This will create a routine of frequent improvement of the model and in consequence, the predictions.

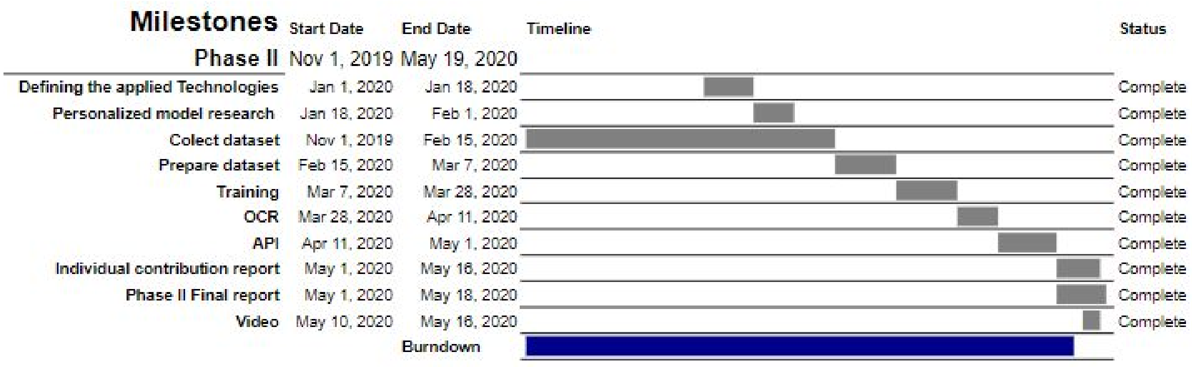
Finally, the specificity of the demand required a considerable amount of experimenting and understanding the processes and their result on the go. This created a few blocking points in the development, but it was important for the teams learning curve.

Project Schedule

The chart below represents the path followed until the completion of this project.



Milestones	Start Date	End Date	Status
Phase II	Sep 1, 2019	Dec 20, 2019	
Identify business problem	Sep 1, 2019	Sep 21, 2019	Complete
Validation Interviews	Sep 22, 2019	Oct 19, 2019	Complete
Business information gathering	Sep 22, 2019	Oct 19, 2019	Complete
Requirements gathering	Oct 20, 2019	Nov 2, 2019	Complete
Defining the scope	Nov 9, 2019	Nov 16, 2019	Complete
Solution options research	Nov 17, 2019	Nov 30, 2019	Complete
Technologies research	Nov 17, 2019	Nov 30, 2019	Complete
Individual contribution report	Dec 1, 2019	Dec 7, 2019	Complete
Phase I report	Dec 8, 2019	Dec 20, 2019	Complete
		Burndown	



Milestones	Start Date	End Date	Status
Phase II	Nov 1, 2019	May 19, 2020	
Defining the applied Technologies	Jan 1, 2020	Jan 18, 2020	Complete
Personalized model research	Jan 18, 2020	Feb 1, 2020	Complete
Colect dataset	Nov 1, 2019	Feb 15, 2020	Complete
Prepare dataset	Feb 15, 2020	Mar 7, 2020	Complete
Training	Mar 7, 2020	Mar 28, 2020	Complete
OCR	Mar 28, 2020	Apr 11, 2020	Complete
API	Apr 11, 2020	May 1, 2020	Complete
Individual contribution report	May 1, 2020	May 16, 2020	Complete
Phase II Final report	May 1, 2020	May 18, 2020	Complete
Video	May 10, 2020	May 16, 2020	Complete
		Burndown	

Gray represents the completed task.

Green represents the tasks in progress.

Dark red represents the tasks remaining.

Dark Blue represents the burn-down from the start period to the end.

References

Accuscience. n.d. *Medical, Surgical & Scientific Products & Services | Accuscience Ireland*. [online] Available at: <https://www.accuscience.ie/> [Accessed 19 May 2020].

AlexeyAB, 2019. *Converting Yolo V3 Models To Tensorflow And Openvino(IR) Models*. [online] GitHub. Available at: <https://github.com/AlexeyAB/darknet/wiki/Converting-Yolo-v3-models-to-TensorFlow-and-OpenVINO(IR)-models> [Accessed 4 May 2020].

Brownlee, J., 2020. *A Gentle Introduction To Object Recognition With Deep Learning*. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/object-recognition-with-deep-learning/> [Accessed 1 May 2020].

Brownlee, J. (2019). *14 Different Types of Learning in Machine Learning*. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/types-of-learning-in-machine-learning/ [Accessed 19 Dec. 2019].

Brownlee, J., 2020. *Overfitting And Underfitting With Machine Learning Algorithms*. [online]
Machine Learning Mastery. Available at: <https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/> [Accessed 2 May 2020].

Ciresan, D., Meier, U., Masci, J., Gambardella, L. and Schmidhuber, J., 2011. *Flexible, High Performance Convolutional Neural Networks For Image Classification*. [online] People.idsia.ch. Available at: <http://people.idsia.ch/~juergen/ijcai2011.pdf> [Accessed 27 April 2020].

Docs.nvidia.com. 2019. *Cudnn Developer Guide :: NVIDIA Deep Learning SDK Documentation*. [online] Available at: <https://docs.nvidia.com/deeplearning/sdk/cudnn-developer-guide/index.html> [Accessed 30 April 2020].

Deeplearning.net. (2019). *Welcome — Theano 1.0.0 documentation*. [online] Available at: http://deeplearning.net/software/theano/ [Accessed 19 Dec. 2019].

Frank, E. and Bouckaert, R., 2020. [online] Link.springer.com. Available at: <https://link.springer.com/content/pdf/10.1007%2F11871637_49.pdf> [Accessed 3 May 2020].

Géron, A., 2019. *Hands-On Machine Learning With Scikit-Learn, Keras, And Tensorflow*. 2nd ed. Sebastopol, CA: O'Reilly Media, Inc., pp.471, 477.

GitHub. n.d. *Alexeyab/Darknet*. [online] Available at: <https://github.com/AlexeyAB/darknet/blob/master/README.md> [Accessed 2 May 2020].

GitHub. (2019). *tesseract-ocr/docs*. [online] Available at: https://github.com/tesseract-ocr/docs/blob/master/tesseracticdar2007.pdf [Accessed 19 Dec. 2019].

GitHub. n.d. Tesseract-Ocr/Tesseract. [online] Available at: <https://github.com/tesseract-ocr/tesseract> [Accessed 3 May 2020].

Goodfellow, I., Benjio, Y. and Courville, A., 2016. *Deep Learning*. [online] Deeplearningbook.org. Available at: <http://www.deeplearningbook.org> [Accessed 22 April 2020].

Grevelink, E., 2017. *A Closer Look At Object Detection, Recognition And Tracking*. [online] Intel. Available at: <https://software.intel.com/en-us/articles/a-closer-look-at-object-detection-recognition-and-tracking> [Accessed 27 April 2020].

Guo, H. and Viktor, H., 2004. Learning from imbalanced data sets with boosting and data generation. *ACM SIGKDD Explorations Newsletter*, 6(1), pp.30-39.

Guy, T., n.d. *Theaiguyscode - Overview*. [online] GitHub. Available at: <https://github.com/theAIGuysCode> [Accessed 1 May 2020].

Help.github.com. 2020. *Conditions For Large Files - Github Help*. [online] Available at: <https://help.github.com/en/github/managing-large-files/conditions-for-large-files> [Accessed 8 May 2020].

Juanola, M., 2019. *Speed Traffic Sign Detection On The CARLA Simulator Using YOLO*. [online] Repositori.upf.edu. Available at:

<https://repositori.upf.edu/bitstream/handle/10230/42548/Sanchez_2019.pdf?sequence=1&isAllowed=y> [Accessed 3 May 2020].

Keras.io. (2019). *Home - Keras Documentation*. [online] Available at: https://keras.io/ [Accessed 20 Dec. 2019].

Kriesel, D., 2005. *A Brief Introduction To Neural Networks*. [online] Dkriesel.com. Available at: <http://www.dkriesel.com/_media/science/neuronalenetze-en-zeta2-2col-dkrieselcom.pdf> [Accessed 19 April 2020].

Medium. (2019). *Deep Learning Frameworks: Tensorflow, Theano, Keras and popularly used libraries*. [online] Available at: https://medium.com/alumnaiacademy/deep-learning-frameworks-7fb26b867807 [Accessed 19 Dec. 2019].

Medium. (2019). *Top 5 Machine Learning Libraries*. [online] Available at: https://blog.bitsrc.io/top-5-javascript-machine-learning-libraries-604e52acb548 [Accessed 19 Dec. 2019].

Ocr.space. (2019). *Free OCR API*. [online] Available at: http://ocr.space/OCRAPI [Accessed 19 Dec. 2019].

Office Timeline. (2019). *Excel Gantt chart tutorial + Free Template + Export to PPT*. [online] Available at: https://www.officetimeline.com/make-gantt-chart/excel#tutorial-auto [Accessed 19 Dec. 2019].

Opencv.org. (2019). *About*. [online] Available at: https://opencv.org/about/ [Accessed 19 Dec. 2019].

Pytorch.org. (2019). *PyTorch*. [online] Available at: https://pytorch.org/ [Accessed 20 Dec. 2019].

Redmon, J., Divvala, S., Girshick, R. and Farhadi, A., 2016. You Only Look Once: Unified, Real-Time Object Detection. [online] arXiv.org. Available at: <https://arxiv.org/abs/1506.02640> [Accessed 1 May 2020].

Redmon, J. and Farhadi, A., 2018. *Yolov3: An Incremental Improvement*. [online] Pjreddie.com. Available at: <https://pjreddie.com/media/files/papers/YOLOv3.pdf> [Accessed 2 May 2020].

Sathya, R. and Abraham, A., 2013. Comparison of Supervised and Unsupervised Learning Algorithms for Pattern Classification. *International Journal of Advanced Research in Artificial Intelligence*, 2(2).

Schmidhuber, J., 2015. *Deep Learning*. [online] Scholarpedia. Available at: <http://www.scholarpedia.org/article/Deep_Learning> [Accessed 27 April 2020].

Scikit-learn.org. (2019). *scikit-learn: machine learning in Python — scikit-learn 0.22 documentation*. [online] Available at: https://scikit-learn.org/stable/ [Accessed 20 Dec. 2019].

Shalev-Shwartz, S. and Ben-David, S., 2014. [online] Available at: <https://www.cse.huji.ac.il/~shais/UnderstandingMachineLearning/> [Accessed 1 May 2020].

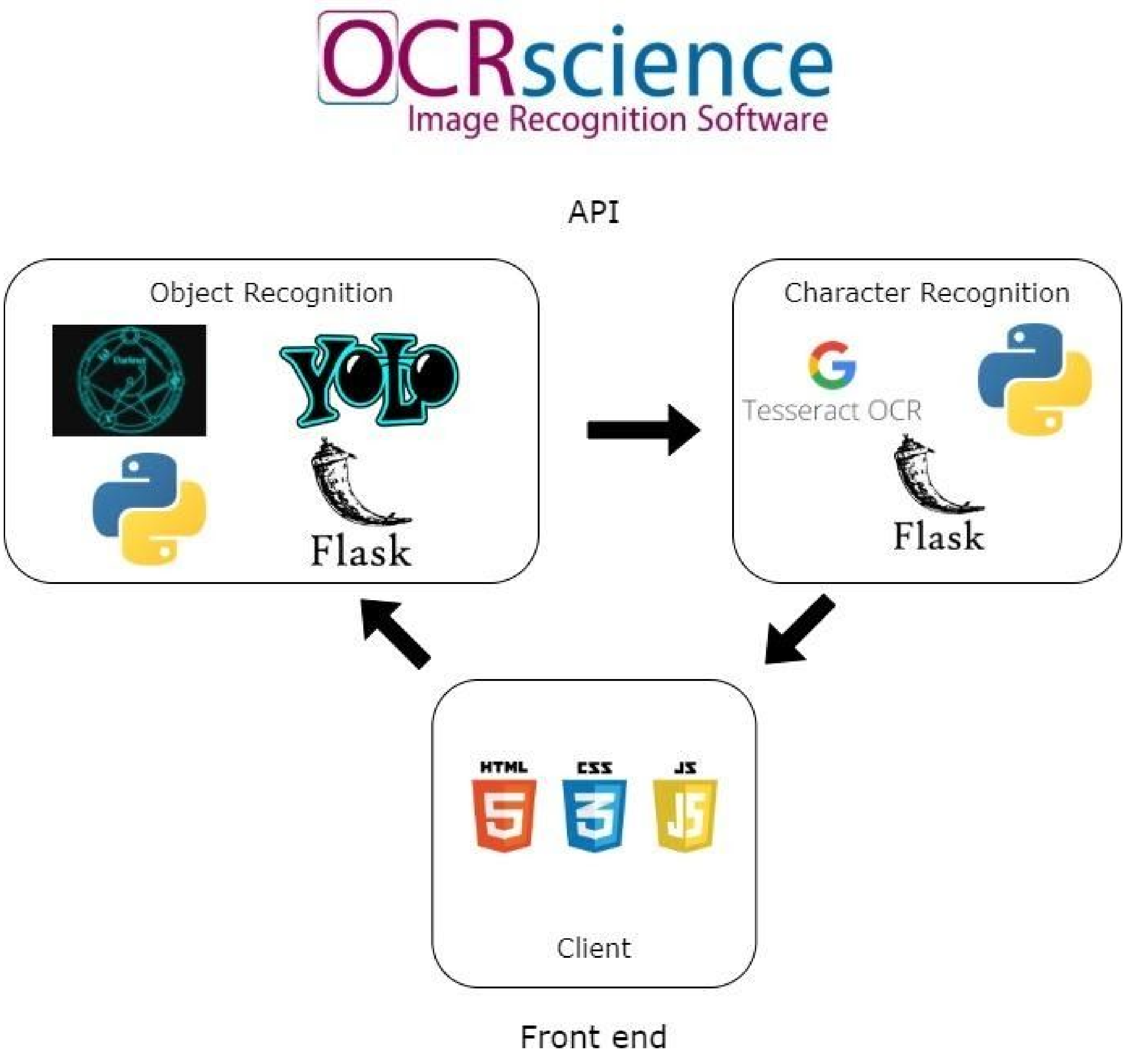
Slant, 5., libraries?, W. and FineReader, A. (2019). *Slant - 5 Best OCR libraries as of 2019*. [online] Slant. Available at: https://www.slant.co/topics/2579/~best-ocr-libraries [Accessed 19 Dec. 2019].

TensorFlow. (2019). *TensorFlow*. [online] Available at: https://www.tensorflow.org/ [Accessed 19 Dec. 2019].

tessdoc. n.d. *Tesseract Documentation*. [online] Available at: <https://tesseract-ocr.github.io/tessdoc/Home.html> [Accessed 4 May 2020].

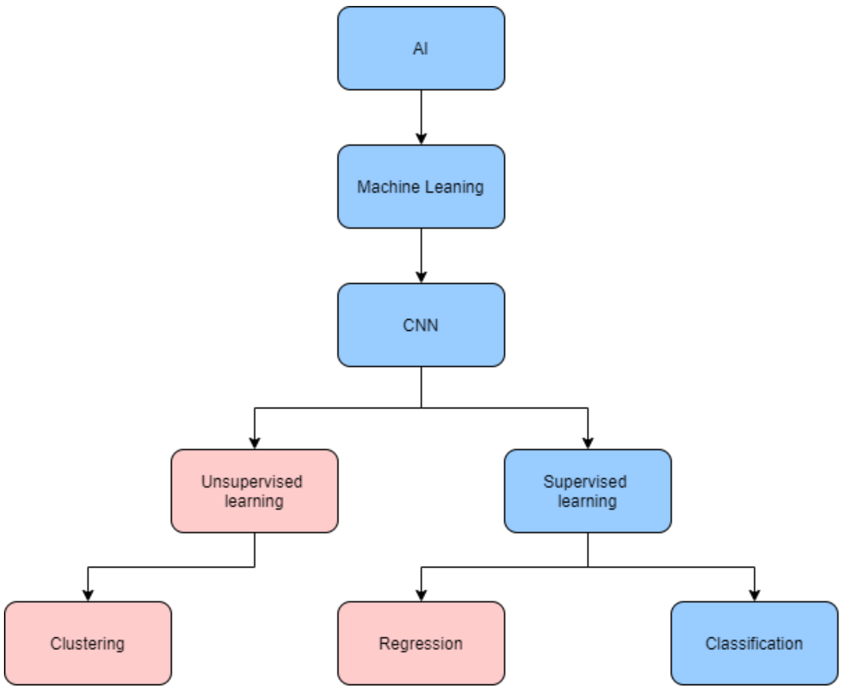
Zhu, C. and Wang, Z., 2017. Entropy-based matrix learning machine for imbalanced data sets. *Pattern Recognition Letters*, 88, pp.72-80.

Appendix A


OCRscience
Image Recognition Software
API
Object Recognition
Flask
Character Recognition
Tesseract OCR
Flask
HTML
CSS
JS
Client
Front end

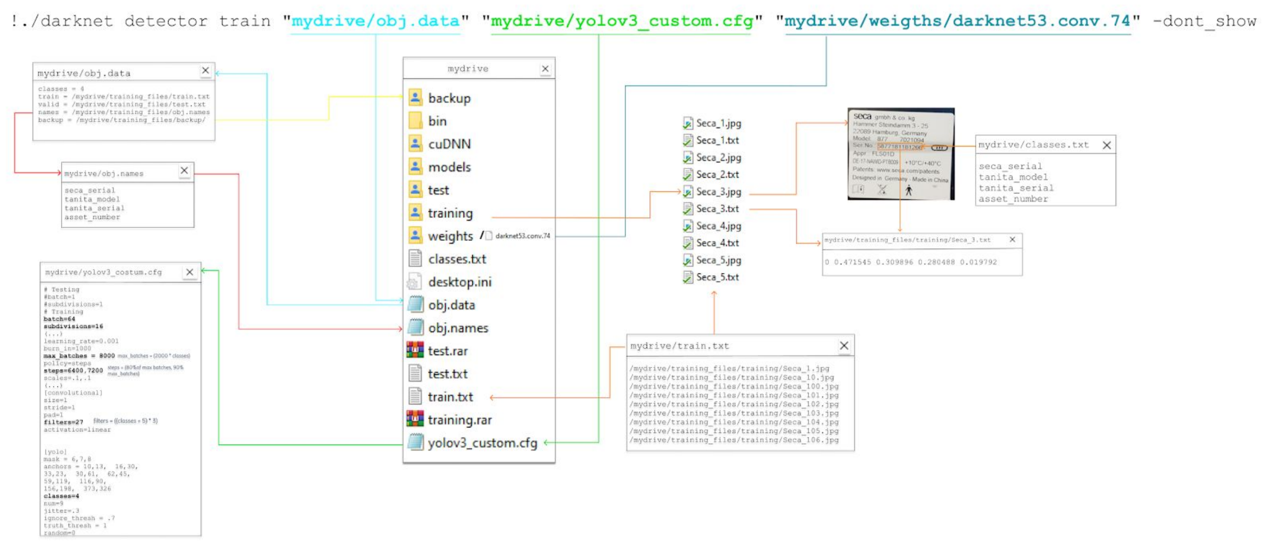
Appendix B

Machine Learning Hierarchy



Appendix C

Training Architecture Graph



```
!./darknet detector train "mydrive/obj.data" "mydrive/yolov3_custom.cfg" "mydrive/weigths/darknet53.conv.74" -dont_show
```

mydrive/obj.data

```
classes = 4
train = /mydrive/training_files/train.txt
valid = /mydrive/training_files/test.txt
names = /mydrive/training_files/obj.names
backup = /mydrive/training_files/backup/
```

mydrive/obj.names

```
seca_serial
tanita_model
tanita_serial
asset_number
```

mydrive/yolov3_costum.cfg

```
# Testing
#batch=1
#subdivisions=1
# Training
batch=64
subdivisions=16
(...)
learning_rate=0.001
burn_in=1000
max_batches = 8000  max_batches = (2000 * classes)
policy=steps
steps=6400,7200  steps = (80%of max batches, 90% max_batches)
scales=.1,.1
(...)
[convolutional]
size=1
stride=1
pad=1
filters=27  filters = ((classes + 5) * 3)
activation=linear


[yolo]
mask = 6,7,8
anchors = 10,13,  16,30,
33,23,  30,61,  62,45,
59,119,  116,90,
156,198,  373,326
classes=4
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=0
```

mydrive/classes.txt

```
seca_serial
tanita_model
tanita_serial
asset_number
```

mydrive/training_files/training/Seca_3.txt

```
0 0.471545 0.309896 0.280488 0.019792
```

mydrive/train.txt

```
/mydrive/training_files/training/Seca_1.jpg
/mydrive/training_files/training/Seca_10.jpg
/mydrive/training_files/training/Seca_100.jpg
/mydrive/training_files/training/Seca_101.jpg
/mydrive/training_files/training/Seca_102.jpg
/mydrive/training_files/training/Seca_103.jpg
/mydrive/training_files/training/Seca_104.jpg
/mydrive/training_files/training/Seca_105.jpg
/mydrive/training_files/training/Seca_106.jpg
```

Marcela Souza
Ana Bazerque
Davi Moraes

Appendix D

Java code to create Test.txt and Test.txt

```
package renameFiles;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileWriter;

import java.io.FilenameFilter;

import java.io.IOException;

import java.util.Scanner;


public class renameFiles {

  public static void main(String[] args) {

    //read all files from a directory

    //read only jpg format

    //create new txt file named train.txt

    //add my personal path

      File[] listOfFiles = readFromFilterOnly("C:\\Users\\ronil\\Google
Drive\\MAD_Code\\training_files\\img_B", ".jpg");

      writeToFile("C:\\Users\\ronil\\Google
Drive\\MAD_Code\\training_files\\train.txt", listOfFiles, "data/obj/");

      ReadFile("C:\\Users\\ronil\\Google
Drive\\MAD_Code\\training_files\\train.txt");

  }

      /*retrieve only jpg file from a given directory

      * First parameter the path from where the it will read all the
files
```

```
     * Second parameter sets the type of file extension it will be
reading from
     */
    public static File[] readFromFilterOnly(String dirName, String
filter){
    File dir = new File(dirName);
    return dir.listFiles(new FilenameFilter() {
      public boolean accept(File dir, String filename){
        return filename.endsWith(filter);
      }
    });
  }
     /* Create a txt file
     * First parameter is the path it set the location where the file
will be created
     * Second parameter is the name of the file to be created
     * Third is the extension of the file - It will set the type of
the file
     * return the patch of the file just created
     */
   public static String createTxtFile(String path, String fileName,
String extension) {
    try {
      File myObj = new File(path +"\\"+fileName+"."+extension);
      if (myObj.createNewFile()) {
        System.out.println("File created: " + myObj.getName());
                              System.out.println("Absolute path: " +
myObj.getAbsolutePath());
        return myObj.getAbsolutePath();
      } else {
        System.out.println("File already exists.");
      }
    } catch (IOException e) {
      System.out.println("An error occurred.");
      e.printStackTrace();
    } return null;
  }
      /*write into a file
      * First parameter is the file we will use to write data on
      * Second is a array of files
      * Third add a prefix to the file name
      */
      public    static    void    writeToFile(String    fileName,    File[]
listOfFiles, String prefix) {
      try {
      FileWriter myWriter = new FileWriter(fileName);
      for (int i = 0; i < listOfFiles.length; i++) {
        myWriter.write(prefix + listOfFiles[i].getName()+"\n");
      }
      myWriter.close();
      System.out.println("Successfully wrote to the file.");
    } catch (IOException e) {
      System.out.println("An error occurred.");
      e.printStackTrace();
    }
  }
   // Reads each entry from a given file
      public static void ReadFile(String fileName) {
```

```
    try {
     File myObj = new File(fileName);
     Scanner myReader = new Scanner(myObj);
     while (myReader.hasNextLine()) {
       String data = myReader.nextLine();
       System.out.println(data);
     }
     myReader.close();
   } catch (FileNotFoundException e) {
     System.out.println("An error occurred.");
     e.printStackTrace();
   }}
    //retrieve all files from a given directory
    public static File[] findAll (String dirName) {
    File folder = new File(dirName);
    File[] listOfFiles = folder.listFiles();
    return listOfFiles;
     }
/*print files from a file array
 * Loop through a given files array and print if is file or directory
 */
public static void readFromListOfFiles(File[] listOfFiles ) {
  for (int i = 0; i < listOfFiles.length; i++) {
    if (listOfFiles[i].isFile()) {
      System.out.println("File " + listOfFiles[i].getName());
    } else if (listOfFiles[i].isDirectory()) {
      System.out.println("Directory " + listOfFiles[i].getName());
```

```
          }
        }
      }
    }
```

Appendix E

The snippet below shows the Python implementation of using the device's camera to get the input to the program. This is a feature to be further developed.

```
from IPython.display import display, Javascript
from google.colab.output import eval_js
from base64 import b64decode
def take_photo(filename='photo.jpg', quality=0.8):
  js = Javascript('''
    async function takePhoto(quality) {
      const div = document.createElement('div');
      const capture = document.createElement('button');
      capture.textContent = 'Capture';
      div.appendChild(capture);
      const picture = document.createElement('picture');
      picture.style.display = 'block';
      document.body.appendChild(div);
      div.appendChild(picture);
      return picture.toDataURL('image/jpeg', quality);
    }
    ''')
  display(js)
  data = eval_js('takePhoto({})'.format(quality))
  binary = b64decode(data.split(',')[1])
  with open(filename, 'wb') as f:
    f.write(binary)
  return filename
```

Appendix F

Interviews Documentation

Interview Summary			**Numb.:**01
System: Scales Calibration			
Interviewee: Tiarnán McCarville			
Interviewer: Marcela Souza			
Date:15/09/2019	**Time:**7pm	**Duration:** 30min	**Place:**Coffee shop
Purpose of the interview: Identify an issue and the possibility of a project			
Agenda: ● Understand his workplace and attributions at Accuscience			
Interview highlights:			

First time Tiarnan mentioned his work, he was unhappy with the amount of time wasted filling in the paperwork. He thinks that maybe a simple barcode reader could save him a lot of time. **Questions:** 1. *What is exactly that you work with?* Tiarnan works calibrating scales. 2. *How would a bar code reader be able to help?* The bar code would contain the scales calibration information. To update the calibration he would only need to fill in the form online instead of the paperwork afterwards. 3. *What would be useful features of the desired system?* The barcode on the scale would be read on the company's phone where he could update its information. 4. *What kind of equipment are included in the calibrations scope?* Tiarnan only calibrates scales although his company has many other equipment such as MR machines **Notes:**

Interview Summary			**Numb.:**02
System: Scales Calibration			
Interviewee: Mr. Tiarnán McCarville			
Interviewer: Marcela Souza, Ana Bazerque, Davi Moraes			
Date: 21/09/2019	**Time:** 8pm	**Duration:** 30min	**Place:** Skype
Purpose of the interview: Explore more details about the issue			
Agenda:			

- Introduce Tiarnan to the team
- Identify a possible manager
- Ask details about workflow

Interview highlights:

Mr. McCarville will contact his manager and explain to him about our project intentions

Questions:

1. *What happens after you give back the paperwork?*
   He returns the paperwork into the office and the documents are scanned but has no information about what happens afterwards.

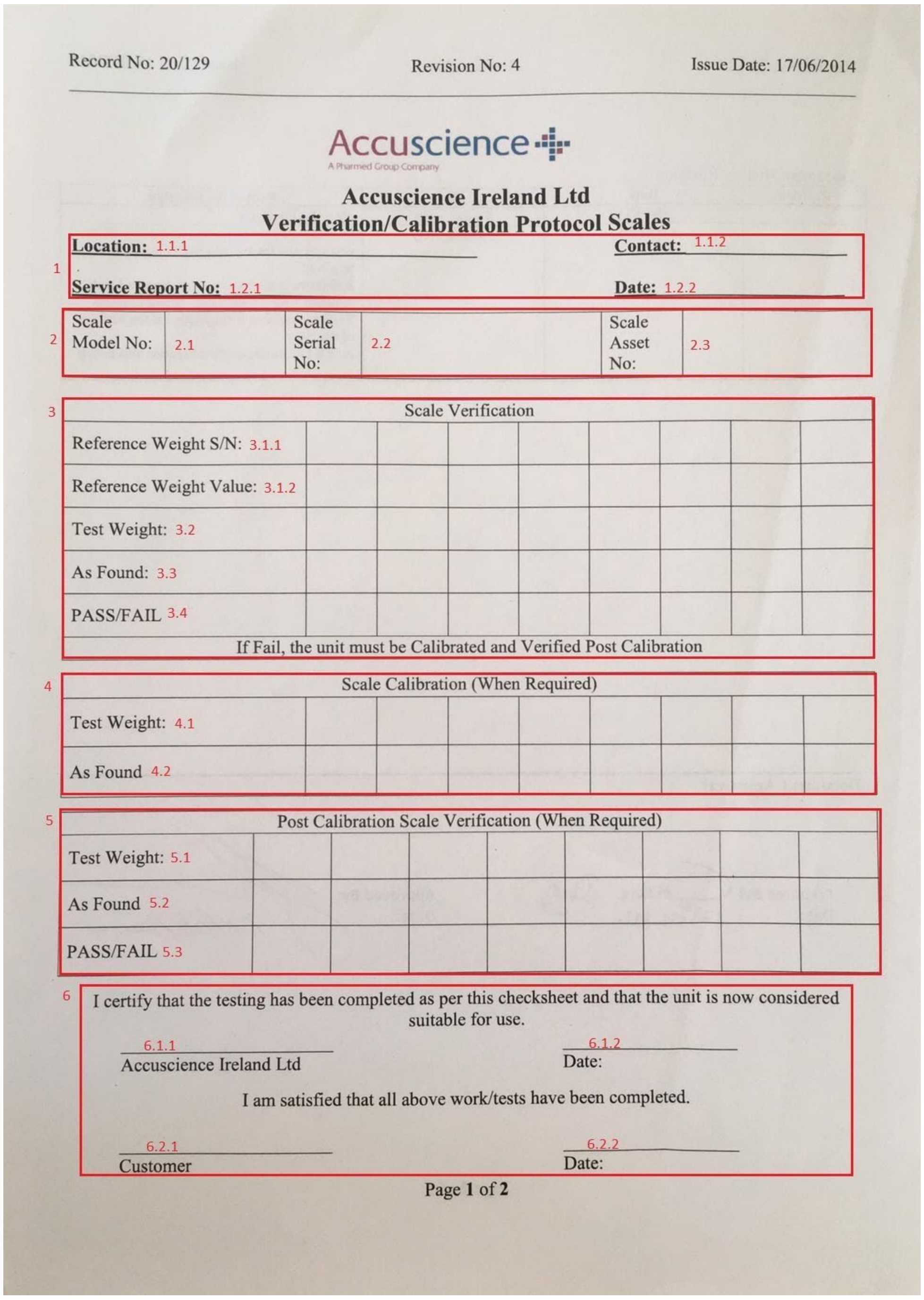
2. *Do you think your boss would be open to talk to us?*
   He might be open to talk to us but, he wants to talk to him and his manager first.

Notes:
Tiarnan suggested that he could be a middle man and introduce us by email containing our college proposition and send it to him forward to his manager.

Interview Summary			**Numb.:**03
System: Scales Calibration			
Interviewee: Ms. Matt McParland			
Interviewer: Marcela Souza, Ana Bazerque, Davi Moraes			
Date: 18/10/2019	**Time:** 2pm	**Duration:** 1h	**Place:** Coffee shop
Purpose of the interview: Introduce the team to Mr. McParland and understand the system requirements			
Agenda: ● Calibration workflow ● Currently system functionalities ● Profiles involved in the workflow			

● Calibration requirements ● Equipement
Questions: 1. *What is the calibration workflow from end to end and what are the documents necessary to the process?* 2. *What kind of system do you currently have and its functionalities?* 3. *How many people are involved in the end to end process of the calibration and what are their roles in this process?* 4. *How does the engineer know when the calibration is due? Who schedules new calibration and how?* 5. *Besides scales what kind of equipment is calibrated by Accuscience?* **Notes:** Mr. McParland provided the following documentation for analyses ● System requirements ● Calibration form

Appendix G



Record No: 20/129 Revision No: 4 Issue Date: 17/06/2014

Accuscience Ireland Ltd

Verification/Calibration Protocol Scales

1

Location: 1.1.1 **Contact:** 1.1.2

Service Report No: 1.2.1 **Date:** 1.2.2

2

Scale Model No:	2.1	Scale Serial No:	2.2	Scale Asset No:	2.3

3

Scale Verification

Reference Weight S/N: 3.1.1								
Reference Weight Value: 3.1.2								
Test Weight: 3.2								
As Found: 3.3								
PASS/FAIL 3.4								

If Fail, the unit must be Calibrated and Verified Post Calibration

4

Scale Calibration (When Required)

Test Weight: 4.1								
As Found 4.2								

5

Post Calibration Scale Verification (When Required)

Test Weight: 5.1								
As Found 5.2								
PASS/FAIL 5.3								

6

I certify that the testing has been completed as per this checksheet and that the unit is now considered suitable for use.

6.1.1
Accuscience Ireland Ltd

6.1.2
Date:

I am satisfied that all above work/tests have been completed.

6.2.1
Customer

6.2.2
Date:

Page **1** of **2**

Accuscience scale calibration form

The purpose of the form is to document what happened during the calibration of the scales. It provides information to both the client and the engineers about how the scale interacted with the different weight classes and provides written proof that the scale was serviced and if it is or is not fit for medical use.

Form fields:

1. **Header**
    - 1.1. Header
        - 1.1.1. Location
          The location of where the inspection will take place
        - 1.1.2. Contact
          The Person that actually schedule the calibration
    - 1.2.
        - 1.2.1. Service Report No
          It's the invoice number
        - 1.2.2. Date
          Day of calibration
2. **Scales tags information** (image x)
    - 2.1. Scale Model No
      Scale Model number found on the scale to be calibrated
    - 2.2. Scale Serial No
      Scale Serial number found on the scale to be calibrated
    - 2.3. Scale Asset No
      Scale Asset number found on the scale to be calibrated.
      Field used only for hospitals. Every hospital has an asset number of each equipment.
      In the case where there is no Scale Asset No a "N/A" is applied instead

3. Scale Verification

3.1. Weights

3.1.1. Reference Weight S/N

Reference Weight Serial Number.

Every engineer has they own set of weights and each of them as their own serial number

3.1.2. Reference Weight Value

The weight value is the mass of the weight in Kg / g. It's correspondent serial number is declared above

3.2. Test Weight

There are three types of group weights: Baby Scales, Adult and Theater / Nappy Scales. Its groups weights are described below at table x

3.3. As Found

It's the weight value found on the scale during calibration

3.4. PASS/FAIL

If the weight value found is the same as the group weight then the scale PASS the test else it FAILS

4. Scale Calibration

4.1. Test Weight

The weight value used on the calibration

4.2. As Found

It's the weight value found on the scale during calibration

5. Post Calibration

5.1. Test Weight

The weight value used on the calibration

5.2. As Found

It's the weight value found on the scale during calibration

5.3. PASS/FAIL

If the weight value found is the same as the group weight then the scale PASS the test else it FAILS

6. **Assignature**

6.1.1. Accuscience Ireland Ltd
Engineer signature

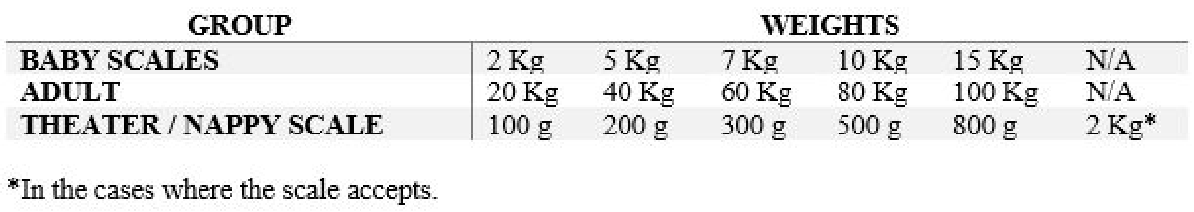
6.1.2. Date
The date of the calibration

6.2.

6.2.1. Customer
Customer signature

6.2.2. Date
The ate of the customer signature

Calibration weight group



GROUP	WEIGHTS					
BABY SCALES	2 Kg	5 Kg	7 Kg	10 Kg	15 Kg	N/A
ADULT	20 Kg	40 Kg	60 Kg	80 Kg	100 Kg	N/A
THEATER / NAPPY SCALE	100 g	200 g	300 g	500 g	800 g	2 Kg*

*In the cases where the scale accepts.

Table x - Calibration weight group

Individual Contributions

Contributor: Ana Bazerque

For the duration of this project I was responsible for the documentation report and, in consequence, I end up playing a Project Manager role, even though the team is responsible for their individual tasks. For the research part, the subject of machine learning was quite new to us, so the plan was to divide and conquer. We split the research between theory, libraries, and how they are applicable for this project. Then we shared our findings among ourselves to make all the decisions. I was in charge of the theoretical side of the research and I was able to give relevant input for the decision making process.

When we entered the practical stage of the development, I was responsible for the first test implementation of the OCR algorithm being used to read one of the first few pictures we had at that point. That was a turning point for us and when we realised it was possible to continue with the plan. The whole group then engaged in the dataset collection and augmenting, when we divided among us the more than 1000 pictures to manually label. And this laborious work had to be repeated for parts of the dataset when the second model was being prepared.

Since we could not work from the same place, we decided that the three of us should try and build the environment to train the custom model. This decision was made based on the fact that we understood this was the core of the system and we should all know the mechanics of it. I made a try with the set up I created for the training, but as I mentioned before, some of the labels were corrupted on the first labeling and the next attempt to train was run on Davi's machine, which was the first successful model trained.

After the first model was trained and we all knew the details of it, we went back to dividing the tasks we had in our hands. While Davi moved forward with the development of the API, Marcela started building the application's front end, I went back to the documentation, to record all that was accomplished and to make sure we have the best and clearest report we could produce as well as making sure our supervisor was aware of our progress.

The learning curve we all went through progressed rather well despite a few stumbles and I am very happy with the results of this project. I believe the success of this development is due to the fact that we were all very determined and each of us were able to complete what the other one lacks in terms of skills.

Contributor: Marcela souza

For the first part of the project, my role was to be the bridge between our group and the cliente. My tasks included coordinating the interviews, exchanging emails and also, document all our interviews with our client Mr. McParland. As a team, we all sat together to create the emails and questions for all interviews that were conducted.

As Ana mentioned we divided the research and my responsibility was to design the charts comparing the Machine learning Libraries and OCR libraries to identify the best solution for our project. After the decision was made on what technologies to use, myself and the group selected some tutorials and blog posts to help assist us on this process as well as documenting the concluding comparisons of the different technologies we used .

The phase of treating the dataset was a group effort. My task on this step was to perform the augmentation of the under-represented classes, label one third of the dataset, improve the Colab step by step and then used this to train our final model.

Through the documentation I was responsible for all the images, graphs, tables and the form explanation. Also, I added a rough description and conclusions of the thought process which was later polished by Ana. My last task in this project was to create the frontend of the API request.

Myself and my team members took an organised and professional approach to this project. We made sure that everyone attended each team meeting and that everyone's voice was heard throughout the decision process, as the individual collaboration during our work was crucial to the success of this project.

Each of us brought to this project our individual skill sets which enriched this project and our bonds. Our unique capabilities were never so well synchronised as in the second phase of this project and I am proud of what we have accomplished together.

Contributor: Davi Moraes

During the first part of the project, I was responsible for researching integration solutions for OCR and Object Detection technologies. During this stage, I made some tests using Google's Firebase platform and also Tesseract.js.

In the second part, we defined Darknet/YOLO as the detection system to be used in the project and the creation of the Darknet model was a result of a collective effort. Like the other team members, I participated in the preliminary research on the basic setup for the creation of the Darknet model, the image selection and the labelling of the images.

Once the first version of the Darknet was done, my first duty was to plan strategies regarding how the API should deal with the images uploaded for detection. Then I made a research on which programming languages and libraries could be used to extract data from the Darknet model detection result. With Python and CV2 defined as the chosen ones, I started to write the entire API using the FLASK framework as the basis for the webserver.

The following step was creating the front-end, of which I wrote the back-in-the-front and the basics of the HTML file. The final visual aspects of the front-end is a result of the improvements made by Marcela.

The whole group was fully engaged in doing their best to carry this project as far as possible and the results achieved are proof of that. We had courage to define as a subject of study a field that was completely unexploited by us, and we got this thing perfectly done.