University of
South-Eastern Norway

www.usn.no

PRH612-1 21V Bachelor´s Thesis

Level measurements with computer vision – a comparison of traditional and machine learning methods

IA6-5-21

Faculty of Technology, Natural Sciences and Maritime Sciences
Campus Porsgrunn

University of
South-Eastern Norway

www.usn.no

Course: PRH612-1 21V Bachelor thesis

Title: Level measurements with computer vision – a comparison of traditional and machine learning methods

This report forms part of the basis for assessing the student's performance in the course.

Project group: PRH612-1 21V Bachelor thesis **Availability:** Open

Group participants: Eirik Døble
Sindre Haugseter
Christian Mikkelsen
Jørgen Sneisen

Supervisor: Ole Magnus Brastein, Nils-Olav Skeie

Project partner: Bouvet

Approved for archiving: ____
(supervisor signature)

Summary: Bouvet initiated a project for creating an automated barista with a robot arm to serve coffee to users. As a part of this they needed a non-intrusive measurement of coffee level in the machine. This need created a foundation for this project. The main objectives were: Create level estimation with Machine Learning, create level estimation with a traditional image processing method, and create an easily accessible user interface to compare machine learning and traditional method.

The level prediction models were created with python and docker hosted on Azure container instances. The website is created with a bootstrap template and hosted on Azure as a Webapp, when a user takes a picture it sends two HTTP calls to the API, which returns the predicted level, and an image representing the way the model interpreted the picture.

The comparison of the two methods shows that under ideal conditions, both methods can be useful as level predictors. However, when the pictures contain more disturbances in the background machine learning is more robust in handling these disturbances.

Preface

Students for this bachelor thesis are on the last semester of the *Computer science and industrial automation* program in USN, campus Porsgrunn. This report is for someone with a basic understanding of modern programming languages and have some interest in machine learning and traditional image processing.

We would like to thank Bouvet ASA for the opportunity to work on their coffee robot project and help from their employees. We also want to thank Geir Arne Evjen for helping us with parts of this project.

Digital tools used in this project are as following; Windows 10, Office 365 pack, Visual studio code, Google colab, Google drive, Microsoft Teams, Git.

Link to the website that has been developed during this project can be found here: https://coffeefinder.azurewebsites.net/

Porsgrunn, 24.05.21

Eirik Døble

[illegible]

[illegible]

[illegible]

Nomenclature

AI – Artificial Intelligence

API – Application Programming Interface

CED – Canny Edge Detection

CNN - Convolutional Neural Network

CPU – Central Processing Unit

FOV – Field of View

GPU – Graphics Processing Unit

HTML5 - HyperText Markup Language 5

HTTP – Hyper Text Transfer Protocol

HTTPS - Hypertext Transfer Protocol Secure

IP – Internet Protocol

JSON – JavaScript Object Notation

ResNet – Residual Neural Network

RGB – Red Green Blue. Color format often used in digital images.

TLS – Transport Layer Security

W3C - World Wide Web Consortium

Contents

1 Introduction

In industry as well as research and commercial sectors, level measurement is important for many applications. The most common and popular measurement technologies are guided radar, ultrasonic, pressure, capacitive and flotation-based sensor principles [1]. Most of these are intrusive and need adaptation to the facility to function properly. A non-intrusive sensor principle is desirable in many applications. One solution that have received significant scientific interest in recent years is the use of advanced machine learning algorithms together with digital cameras [2]. This is a technology with many possible applications, one of which is an non-intrusive level measurement of a substance in a transparent tank.

1.1 Background

Bouvet initiated a project in 2020 to develop and construct a fully automated barista with Artificial Intelligence (AI) and a robotic arm [3]. The part of the project that this report covers aims to deliver a method of level measurement to measure the level of coffee beans in the barista. The goal of the project is to compare a machine learning model against a traditional image processing method in the scope of level measurement. This should be done in a web application. Due to the Corona situation in 2020/21 the coffee robot is inaccessible, and a separate test rig must therefore be made to be able to train and test the program.

1.2 Objectives

1. Level estimation with machine learning.
2. Level estimation with a traditional method.
3. Easily accessible user interface to compare machine learning against the traditional method.

The methods will be implemented and compared in a web application, where a camera takes pictures of a coffee bean tank and sends the pictures for processing to estimate the level in the container. A large part of the work will be to train and adapt the machine learning model, experiments must be planned and performed to collect images that can be used for training data. See Appendix A for a detailed project description (in Norwegian).

1.3 Report Layout

Chapter 2 gives an overview of the systems developed to achieve the results highlighted in this report.

Chapter 3 is about the software architecture and the communication between the modules in the project.

Chapter 4 gives a short overview of machine learning and discusses the implementation of machine learning in the project.

Chapter 5 gives a short overview of traditional image processing and the implementation in the project.

Chapter 6 shows the results from experiments done during the project.

Chapter 7 discusses the result from Chapter 6.

Chapter 8 is the conclusion to the project.

2 System overview

This chapter will give an overview of the systems developed to achieve the results highlighted in this report.

2.1 System architecture

The system consists of a website with a node.js backend for serving the content of the website. For performing the predictions there is developed a prediction engine, with an Application Programming Interface (API), this is an interface between the website showing the results to the user, and the prediction engine for level estimation. The API is accessible through Hyper Text Transfer Protocol (HTTP) requests, and the response is estimated level in percent, and an image that can be shown to the user on the website [4].

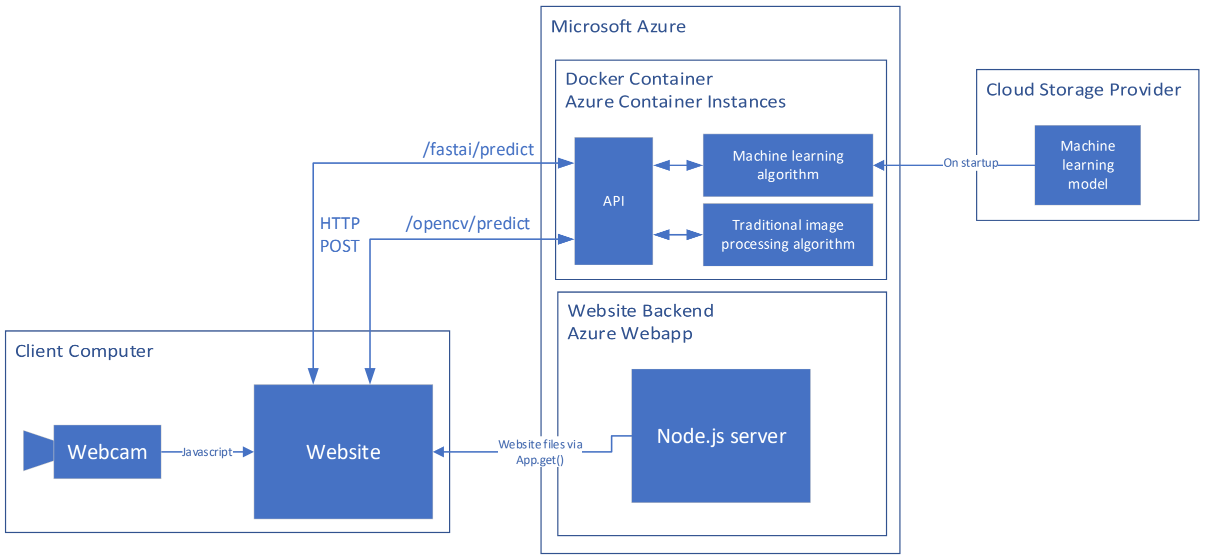


Figure 2.1: Figure shows an overview of the system architecture.

2.2 The Rig

The rig consists of a mounting plate where the camera and the coffee container are mounted to get repeatability in the pictures regarding camera angle. The camera angle was chosen such that the flat top of the coffee container would not disturb the measurement, as shown in Figure 2.2. This is because of the method used for measuring level is effectively measuring the area that is in the cameras field of view (FOV).

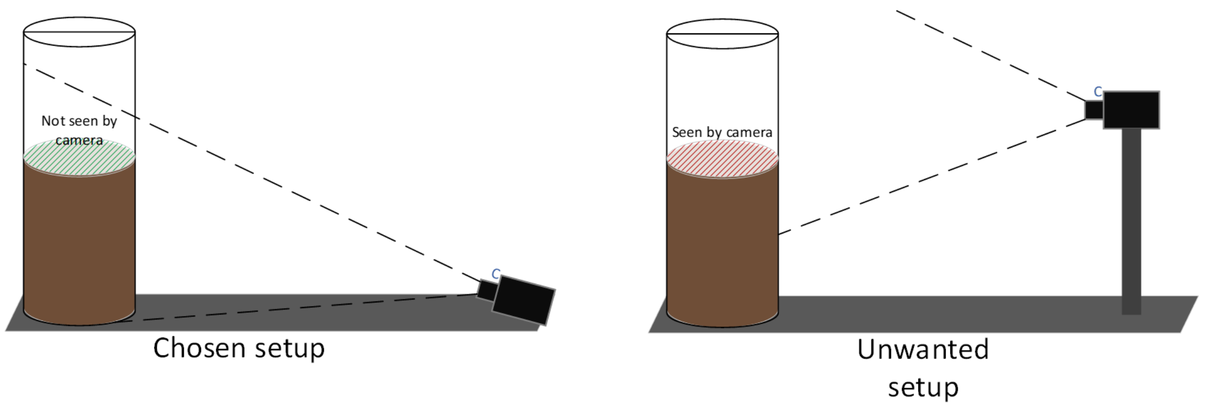


Figure 2.2: Diagram of the rig's setup.

To get enough variety in the background of the pictures the rig was positioned with different walls in the background, and some black paper was used to show that black does not automatically mean 'coffee' for the ML model. The ideal scenarios are a plain light-colored background with little disturbances, even lighting, and little to no reflection.

2.2.1 Perspective distortion

Due to the position of the optical sensor in relation to the tank, as shown in Figure 2.2, there is a distortion of the image. This will affect the measured area by the algorithm. As shown in Figure 2.3, a rectangle of a given size will occupy a bigger pixel area the further down in the image that it is placed. Further also observe that due to lens distortion the rectangles are not completely square. This is likely due to lens distortion. Compensating for these factors would be recommended for future work and is achievable by among others OpenCV technology [5].

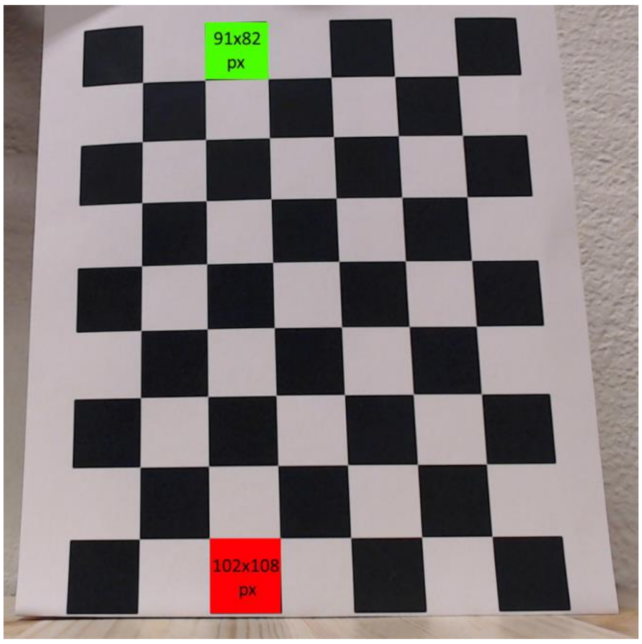


Figure 2.3: Image shows a A4 25x25mm calibration checkerboard [6].

Equation (2.1) shows that the distortion gives a reduction in area measured of ~32% from bottom to top of the image.

$$\frac{Size_{Red} - Size_{Green}}{Size_{Red}} * 100\% = 32.26\% \tag{2.1}$$

2.3 Image storage

The training data is stored in a google drive folder, which enables the images, python files and the ML model file to be accessed in Google Colab. Colab is a free browser-based service, connected to google drive, hosting Jupyter notebooks for running python code and offers free access to powerful computing resources including GPUs [7]. The images and masks are saved in their respective subfolders. Images as masks are connected together by the filename. The images will have name: <filename>.<filetype> while the corresponding mask will have the name <filename>_P.<filetype>.

Images that are taken when predicting is not saved outside of volatile memory.

2.4 Traditional image processing & machine learning

Traditional image processing is an approach to image processing that often involves multiple connected steps, that has to be manually programmed by an expert. It is accurate on specific problems but struggles with changing illumination, noise and overlapping [8].

Machine learning learns features by itself from the training data. If the machine learning model gets enough variation in the training data, it will be robust to noise and varying illumination [8].

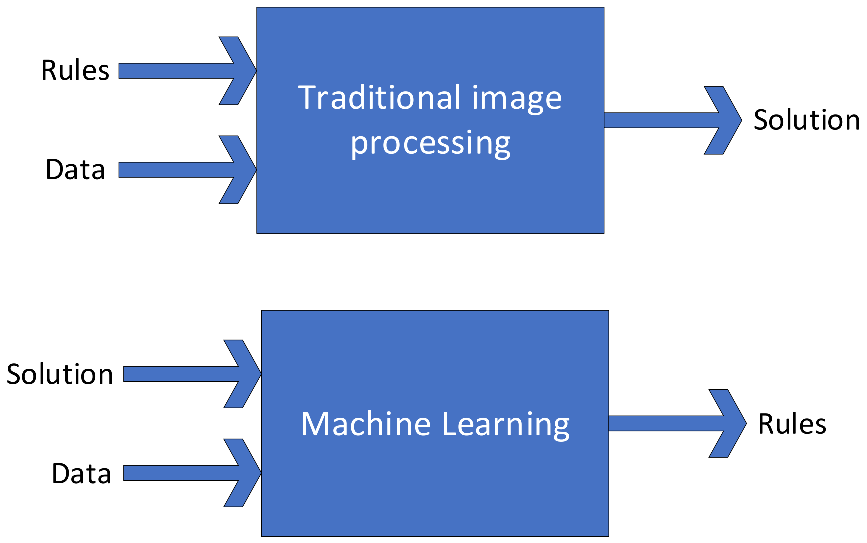


Figure 2.4: Traditional image processing vs Machine learning

As seen in Figure 2.4, traditional image processing requires an expert to manually create the rules for processing the image to get the desired result. This can for example be filtering before doing some operation to extract data from the image. Naturally, some assumptions are made in the creation of these rules. If any external factors cause those assumptions to be invalid, the returned solution might not be correct anymore. Further, in machine learning the algorithm is provided with data and the solution, for example many images of different cats(data) and a label(solution) that states “this is a cat”. After processing this information there is created some rules that can be used in the future to check if another image, not in the initial image set is in fact a cat.

3 Software architecture

To show and compare the two methods there was developed a website that takes pictures through a webcam and sends this to a web Application Programming Interface (API) for performing the predictions. The API is used as an interface between the website and the level estimation models. This results in the level estimation running separate from the website. All the source code is openly available on the projects GitHub repositories [9] [10] [11].

3.1 Hosting on Azure

Both the website and API are hosted on Microsoft's Azure cloud services. The website runs as an Azure App service, which is a fully managed platform that supports node.js, python, .NET and more [12]. The API is run in an Azure Container Instance and receives HTTP calls from the website client [4].

3.2 HTTPS and media devices

All websites should ensure that proper security measures are implemented. The most common way to do this is to use the HTTP protocol with Transport Layer Security (TLS), often referred to as "HTTP over TLS" [13]. This requires a certificate verifying the websites authenticity. This certificate should be created by a verified publisher to avoid security warnings in the client web browser [13].

If a website has implemented Hypertext Transfer Protocol Secure (HTTPS), the HyperText Markup Language 5 (HTML5) standard from World Wide Web Consortium (W3C) makes it possible to capture data from media devices like web-cameras with a simple JavaScript instead of the traditional browser plugins [14]. Media devices is also accessible if the website is accessed locally (localhost).

If a request is made to or coming from an API or client using the HTTP protocol from an external address without the secure layer, the connection is blocked according to the same origin policy [15]. The request from the API host must follow the HTTPS protocol to avoid being blocked by the website, preferably with a valid certificate.

3.3 Website

The website is made with bootstrap, which is a frontend framework for developing responsive websites, and a template from w3Schools [16]. Figure 3.1 shows what the web page looks like.

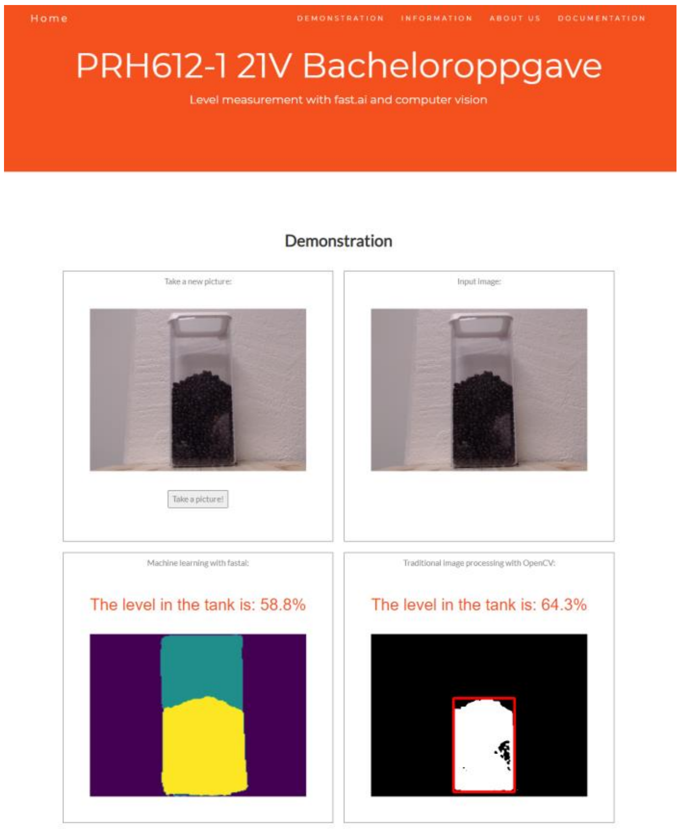

Home
DEMONSTRATION
INFORMATION
ABOUT US
DOCUMENTATION
PRH612-1 21V Bacheloroppgave
Level measurement with fast.ai and computer vision
Demonstration
Take a new picture:
Input image:
Take a picture!
Machine learning with fastai:
Traditional image processing with OpenCV:
The level in the tank is: 58.8%
The level in the tank is: 64.3%

Figure 3.1 Screenshot of the webpage

Nodejs is used as the back-end part for running the server-side logic and serves the website files when the user accesses the website. When a picture is taken on the website for measuring the level of the coffee tank, then two API endpoints are called, one for machine learning estimation and one for traditional level estimation.

The objective of the website is to present the difference between the traditional and machine learning implementations and make it clear to the user on what basis the different models predicted the level in the container. The website uses the clients web camera to capture images of the coffee bean container. As the website needs to access media devices the HTTPS over TLS protocol mentioned in subchapter 3.2 is implemented.

3.4 Application Programming Interface

The API is running in an Azure Container Instance. In this name ‘Container’ refers to a Docker Container. A Docker Container packages up code files and all dependencies in one container, causing the software to run in the same way from one computer to another [17]. The library used to get this to function as a web API is FastAPI [18]. The API has two main endpoints, “https://20.56.206.114[1]/OpenCV/predict” and “https://20.56.206.114[1]/fastai/predict”. Both endpoints take a HTTP POST [4] request with a JavaScript Object Notation (JSON) [19] formatted string containing amongst other things a base64 encoded image [20]. Both endpoints return a JSON object with a graphical representation of the prediction, as well as the level in percent. This will enable the project partner to use the created level estimation methods in the future on their own projects. More information regarding the endpoints is available on the API’s docs site [21].

The API is organized in three main files:

- ApiHandling.py
    - The main file implements the server and calls the other files.
- machineLearning.py
    - File containing the functions for predicting the level with fastai.
- traditional.py
    - File containing the functions for predicting the level with OpenCV.

The flow of the API program is shown in Figure 3.2.

[1] Internet Protocol (IP) address needs to be changed according to the docker containers IP address or Domain Name System name.

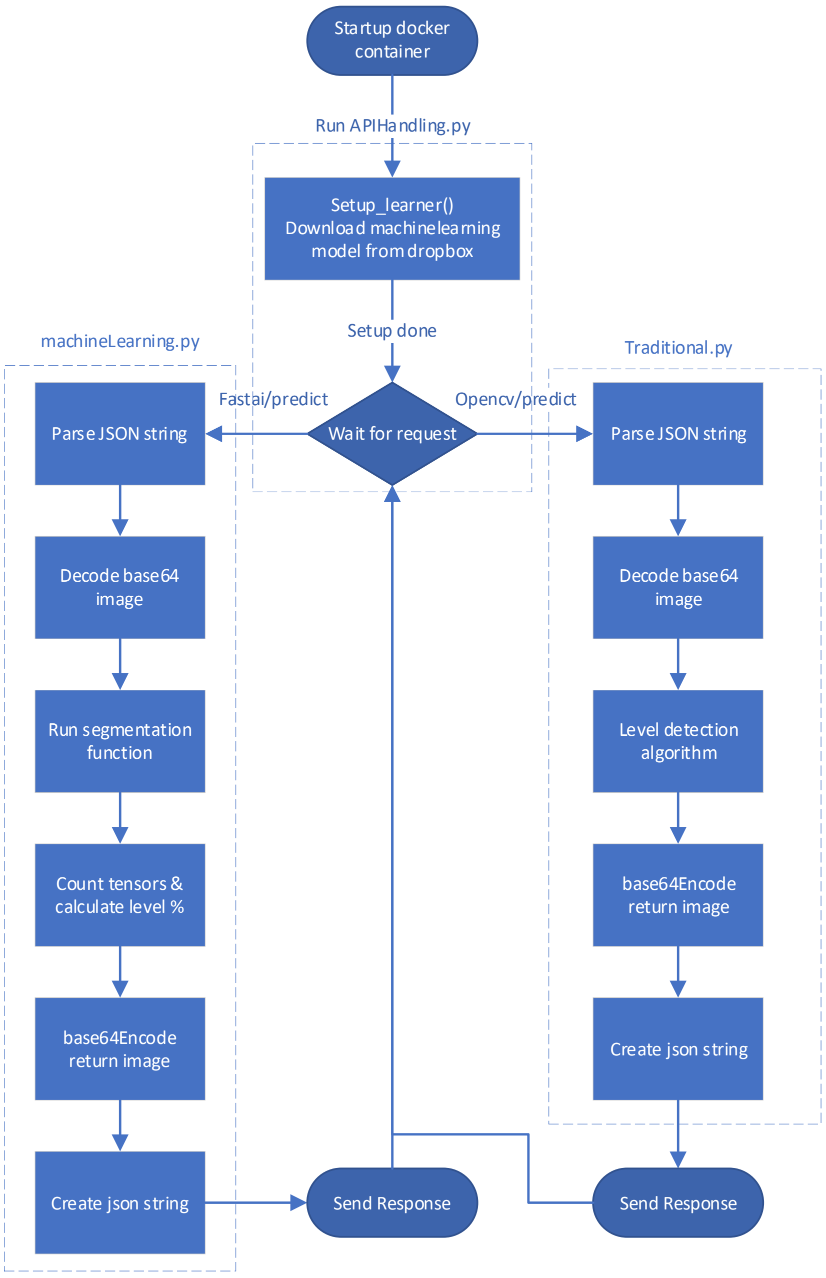


Figure 3.2: Flowcharts of the API program

3.4.1 Docker

Docker containers are used to package up software in such a way that the software runs reliably from one environment to another [17]. This is very similar to virtual machines, but instead of virtualizing hardware, containers virtualize the operating system [17].

Figure 2.1 shows that the API and prediction algorithms are packaged in a Docker container, and hosted on Azure as a container instance [22].

4 Machine learning

Machine learning is a versatile tool that is often used in combination with digital image processing and is highly efficient in patter recognition and working through big data [23]. ML is an interesting framework for preforming level measurement utilizing computer vison.

Subchapter 4.1 gives an overview of computer vision and machine learning. Subchapter 4.2 describes two image processing methods, classification and segmentation. Subchapter 4.3 gives an overview of neural network architectures. Subchapter 4.4 describes the fastai framework and some relevant methods. Subchapter 4.5 shows how it was implemented in the project.

4.1 Computer vision and machine learning (ML)

Computer vision is a field in computer science that is closely linked with artificial intelligence, where the goal is to enable a computer to “see” by identifying and processing images in a way similar to humas [24]. Machine learning is a subset of artificial intelligence as shown in Figure 4.1. A ML model can be trained to preform task by relying on patterns and inference with the help of inductive learning [24] [25] . When combined, the result is a model that can analyses a picture or video and recognize a pattern [24].

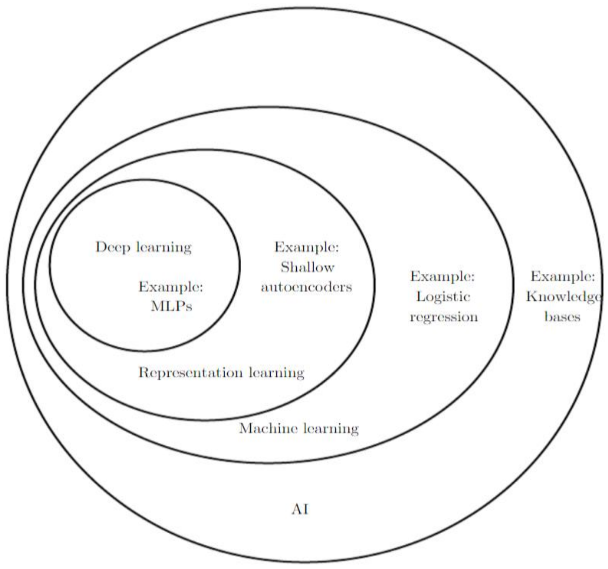


Figure 4.1: A Venn diagram showing the different levels in AI, including an example of an AI technology [2].

In this project computer vison and machine learning are used to find the areas that contains coffee, empty container, or everything else by recognizing the different patterns and colors.

4.2 Classification Vs Segmentation

In this project segmentation was chosen rather than classification. Both segmentation and classification are a part of image processing and machine learning that uses computer vison to train an AI model. Both are used for recognition, but in different ways.

In classification objects in the image are analyzed and labeled by a class or a probability of being in the class [26]. A class can be, for example, a car, a cat, a house, etc. If you give an image of a cat to a ML model, the computer will analyze the image and classify the image as a cat. When training a classification model, the typical method is to put images of the same category in the same folder in the filesystem and name the folder accordingly to each of these categories [27]. When feeding an image to the trained model, the ML model will try to identify each class object [27]. From initial experimentation it is found that classification likely could be used as an alternative to segmentation to solve the problem in this project by classifying different levels in the container. This could be done by feeding the training algorithm images classified as 10%, 20% 30% etc. The model that results from this could most likely only output level measurements for the classes it been trained on, not calculate the level as a continuous size.

Segmentation is a type of classification, but instead of classifying the object in the image, each single pixel needs to be classified [28]. It is often used for tasks that would be manually intensive and needs high pixel-level accuracy [28]. When training a segmentation model, each single object in a picture need to highlight in a different shade to make them recognizable for the machine learning algorithm [27]. This is called a mask. When the trained segmentation model goes through an image it will try to label every pixel in a way that makes it easy to identify which pixel belong to which class, typically also giving them a similar color value for easier visualization [27]. This makes it easy to locate objects and boundaries like lines and curves in an image.

Both methods use an algorithm to train a model from the images. When the training is done, new images is sent to the model and an output is given. This can be seen in Figure 4.2.

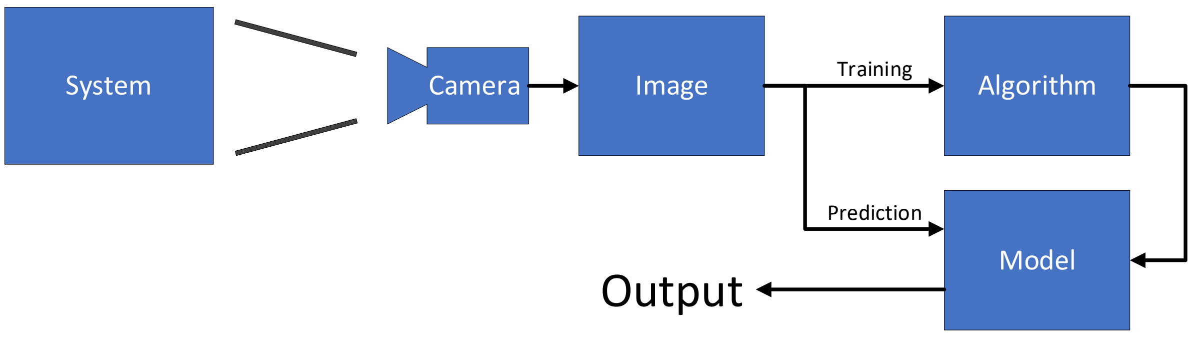


Figure 4.2: Correlation between an ML model and an algorithm

4.3 ResNet

The machine learning model used in this project is built on a Residual Neural Network, or ResNet as its more commonly known. ResNet is an artificial neural network (ANN) based on the constructs known from the pyramidal cells in the cerebral cortex [29]. An example of a neural network is shown in Figure 4.3, this network is defined as a fully connected feed-forward neural network [30]. ResNet is a continuation of a convolutional neural network (CNN) that is popular due to its high effectiveness and practicality [31] [29]. To further improve CNN, ResNet was proposed in 2016, which adds skip connection or shortcuts to jump over some layers [31] [29]. ResNet models typically implements double- or triple- layer skip connections [31] [29]. A weigh matrix can be used to find the skip weights in models known as HighwayNets [31]. The skip connections help to avoids the vanishing gradient problem that occurs in CNN that prevents the neural network from training further because the weight change is vanishingly small [31] [29] [32].

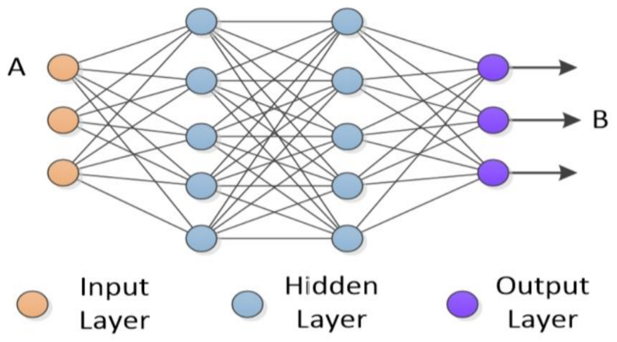


Figure 4.3: A basic representation of a fully connected feed-forward neural network [31].

For a deeper understanding of the mathematics behind ResNet its recommended to read the report “On the Mathematical Understanding of ResNet with Feynman Path Integral” by Y. Minghao , L. Xiu, Z. Yongbing and W. Shiqi [31]. They use the Feynman path integral to give a mathematical expression for the output from the ResNet Model.

The ResNet model used for this project is ResNet-34 which consist of a network with 34 layers. The ResNet-34 model was pre-trained using the ImageNet dataset which consists of millions of images [33]. The model was then re-trained until a satisfactory result was reached at 93 images of the transparent coffee bean container. This process is known as transfer learning [34]. The result from a pre-trained ResNet-34 model can be seen in Figure 4.4. The image shows that the ML model does not know what coffee beans or coffee containers look like.

Larger networks are better at complex problems but are easy to overfit, takes longer to train and use more memory than smaller networks [35]. ResNet-34 was chosen because its performance to accuracy trade-off was satisfactory for the problem. Other neural networks like AlexNet, GoogLeNet, DenseNet and SqueezeNet was tested but based on initial experimentation the ResNet-34 model appears to perform adequately for the task [36].

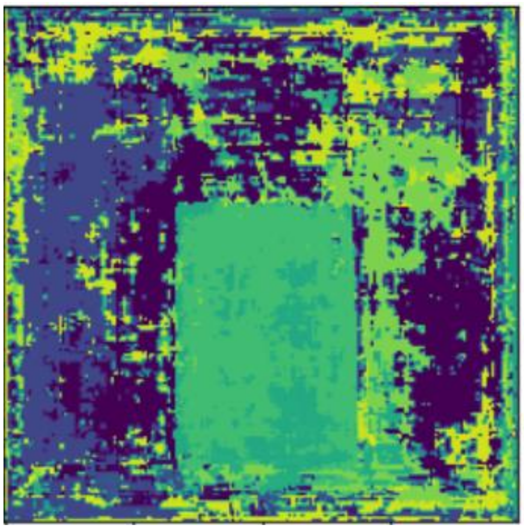


Figure 4.4: ML prediction using the base Resnet34 model without transfer learning.

4.4 Fastai

Fastai is a framework created by the Fast.ai organization. The framework is made with the mindset of providing a practical approach to machine learning, where the idea is to learn by doing practical exercises instead of the traditional theoretical approach where reading and learning comes before experiments [37]. Fastai offers High level API’s, easy to follow code and training videos, making it easier to get started with machine learning [37] [38]. The fastai framework is based on PyTorch and gives the PyTorch library an extra layer of functionality using the API’s [38]. All model training and predictions were done using the fastai framework version 2.2.5 [38].

4.4.1 About PyTorch

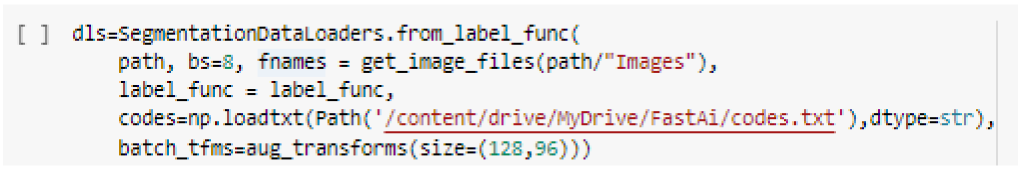
Pytorch is an opensource Python-based scientific computing package mainly developed by Facebook's AI Research lab (FAIR) [39]. It has a specialized library called “torchvision” that handles multiple methods for image processing and supports several architectures for pretrained models. When training models with Pytorch, the results are stored in tensor arrays. In PyTorch these tensors are multi-dimensional arrays optimized for data processing using either a graphical processing unit (GPU) typically is used in graphic cards, or a central processing unit (CPU) which typically is used as the main processor in workstations and servers [40]. The main difference being that GPUs is specialized towards parallel processing and high throughput, while CPUs specializes on low latency. To reduce the time needed for training models, GPUs is the better choice as Pytorch has built in methods optimized for data processing with GPUs. These methods process the tensor data in batches for parallel GPU usage [38].

There are some minimum requirements to start using fastai and Pytorch.

- A collection of reference images
- A collection of masks corresponding to the reference images.
- A list defining class labels.
- A pretrained model, based on a compatible architecture. (ResNet, AlexNet, GoogLeNet etc. [36])

4.4.2 Setup

Some basic steps are required to get fastai up and running. Firstly, fastai needs to organize the reference images, the masks, class labels and the number of images to process at once called batch size and referred to as *bs* in Figure 4.5. These parameters are put into a dataset called a “Dataloader”, using a function called “SegmentationDataLoaders.from_label_func()” which is designed towards segmentation problems [41] [42]. This “Dataloader” is later used as a parameter when creating a ML model learner and is from here on referred to as a data loader, see Figure 4.5. Optionally it is possible to define a transformation function that applies random transformations to the reference images. A ML model with transformations will be able to handle a greater variance in images used as inputs when predicting results [43]. Examples of transformations is: Changing the size, flipping, rotating, or adding gaussian blur.



```
[ ]  dls=SegmentationDataLoaders.from_label_func(
         path, bs=8, fnames = get_image_files(path/"Images"),
         label_func = label_func,
         codes=np.loadtxt(Path('/content/drive/MyDrive/FastAi/codes.txt'),dtype=str),
         batch_tfms=aug_transforms(size=(128,96)))
```

Figure 4.5: Set up the fastai data loader.

The next step is to select a learner for training the model. As this project uses segmentation the “unet_learner” is used instead of the “cnn_learner” commonly used in classification [44]. The “unet_learner” use an CNN architecture called U-Net that specializes towards image segmentation methods like interpreting gradients [45]. The distinctive u-shape of the U-Net architecture is shown in Figure 4.6.

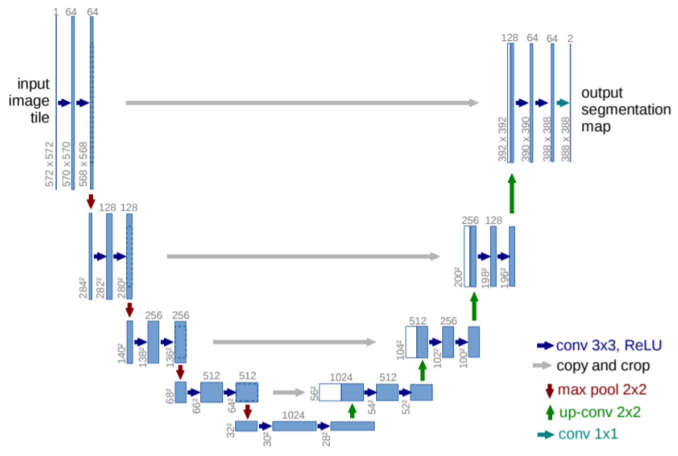
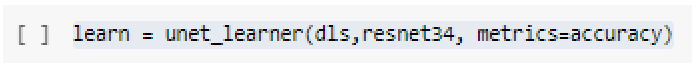


Figure 4.6: The U-Net architecture [45].

Finally, to create a new learner, the “unet_learner” function takes the data loader, a pretrained neural net model, and metrics as inputs, see Figure 4.7. Since classes are predefined in the data loader the learner uses an ML concept called supervised training, meaning that the learner references predefined classes and weighs similar patterns accordingly. The neural net model used for training this projects ML model is ResNet34 which is described in chapter 4.3.



```
[ ]  learn = unet_learner(dls,resnet34, metrics=accuracy)
```

Figure 4.7: Defining the fastai learner.

4.4.3 Handling gradients

Training a model using segmentation with U-Net involves interpreting gradients from tensors. PyTorch uses the class “autograd” to find and calculate these gradients. Mathematically the “autograd” class is just a general mathematical function using forward and backwards propagation algorithms, also called forward and backward pass algorithms, to numerically calculate the training-weight parameters [46] [47]. In other words, “autograd” is a Jacobian-vector product computing engine [46] [47].

Forward propagation involves computing the error with respect to the input training sample by moving through the ML models layers, with the output being the prediction which is used as an input for a loss function calculating the error between predicted and expected output [48] [49]. Back propagation traverses backward through the ML model layers calculating the error deltas or gradient from the output using the chain rule, with the output being the optimizing model weights, effectively making the weights dynamic [46]. Figure 4.8 shows a simplified diagram visualizing the forward and back propagation process.

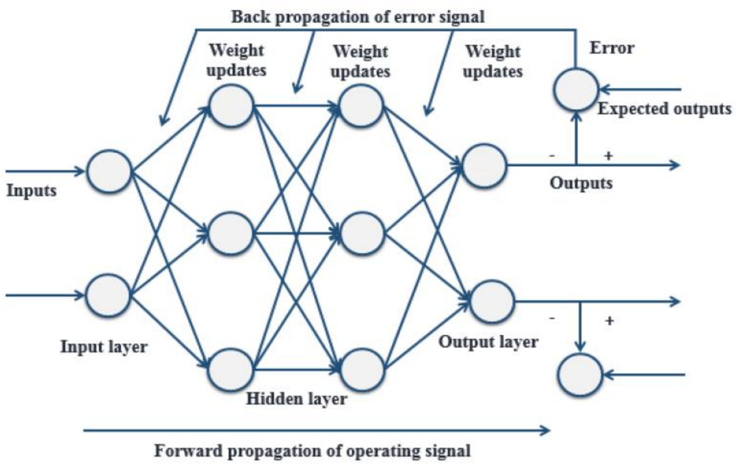


Figure 4.8 A simplified diagram visualizing the forward and back propagation process [50].

4.5 Implementation

The machine learning based approach utilizes image segmentation based on the approach from the fastai framework, see chapter 4.4. The fastai model is trained with an image base of 93 images of the transparent coffee bean tank with varying backgrounds and coffee bean levels. Each image has an associated mask with the same name as the original image with "_P" added at the end. In the mask, coffee is marked with white, empty container are gray and everything else is black. See Figure 4.9.

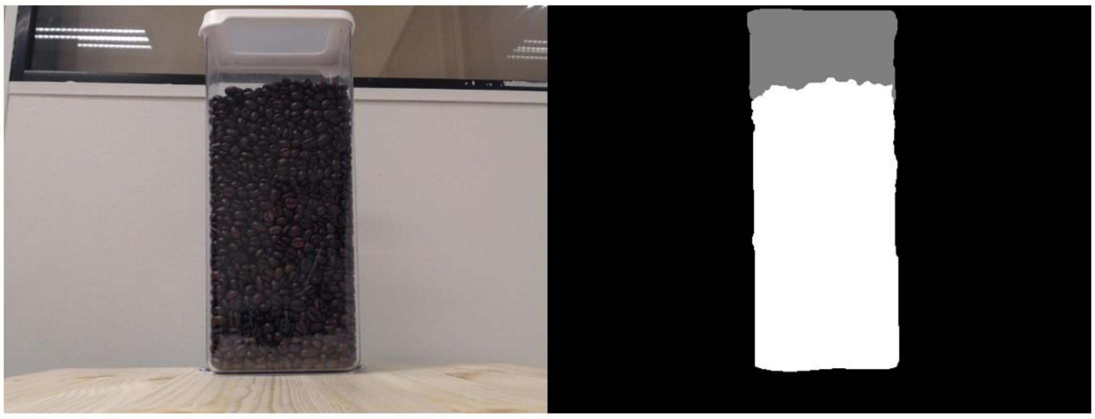


Figure 4.9: Training Picture and corresponding mask.

As mentioned in chapter 4.4.2 setting up a learner to predefining and referencing classes defines the learning process as supervised learning. Preparing the ML model for training is the most time-consuming step of the training process. Creating the masks for each of the images and defining classes was done manually in this project, and with a base of about 100 images the masking process took one person about seven hours. This was done without any experience with image editing. With more experience in image editing this process likely to be more streamlined. The training of the ML model including setting up fastai (using the method described in chapter 4.4) took about 60 minutes for 100 epochs of training. Using a larger batch of reference images would increase the required training time.

The amount of training varies. With too much training the ML model can get overfitted where it will perform good on the training data but get poor results on other data [35]. With too little training it gets underfitted and preforms poor on training data as well as other data [35]. The optimal result is somewhere in between. A graphical representation of this problem can be seen in Figure 4.10.

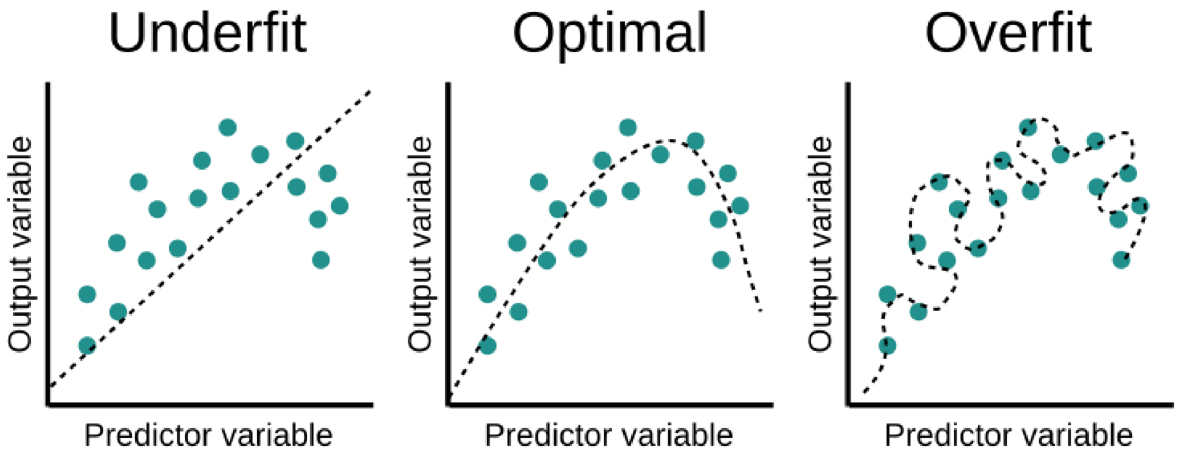


Figure 4.10: Comparison of a underfit, optimal, and overfit ML model [51].

Training a ML model requires a lot of processing power and to avoid the hardware costs and time needed to set up local a computer with fastai and Pytorch, all code for training the ML model used in this project was run on Google Colab [7].

4.5.1 Training:

When the ML model learner is set up as described in chapter 4.4.2 it is ready for training, the training process is simplified by fastai introducing the method *"fit_one_cycle()"* [52]. The method returns loss and accuracy metric in the form of a table, Table 4.1 shows an example of the results, Figure 4.12 and Figure 4.13 shows a plot of loss and accuracy.

Table 4.1: Training data.

Epoch	train_loss	valid_loss	accuracy	time
1	4.35314	4.16750	65.18 %	00:19
60	0.04616	0.03223	98.86 %	00:02

As this project's ML model use segmentation, the loss function compares pixel gradients in the training pictures and masks. This is done with the method described in chapter 4.4.3.

To determine the ML model's performance loss, the predicted output is compared with the target value, and the deviation between these determines the loss values, where a large deviation gives a high loss value [49]. This is visualized for a fully connected feed-forward neural network in Figure 4.11, showing a flowchart of the prediction process, where the target value is based on the reference images and masks. See chapter 4.3 for more information about neural networks and ResNet. As the ML models training iterates through the data, loss and accuracy are calculated for each of these iterations, see Figure 4.12 and Figure 4.13 [49]. Iterations is referred to as epochs in the context of machine learning and is referred to as epochs from this point. When the training reaches a point of around 50 epochs the progression slows down as seen in Figure 4.13. and Figure 4.13.

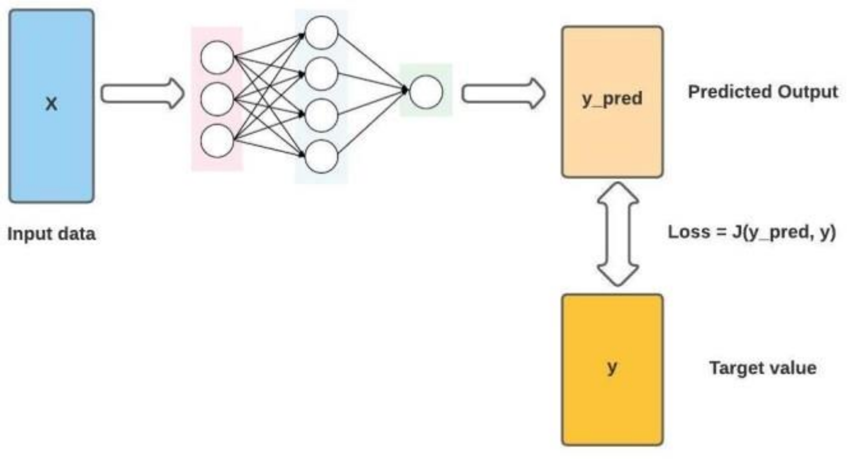


Figure 4.11: Visualizing the prediction process [49]

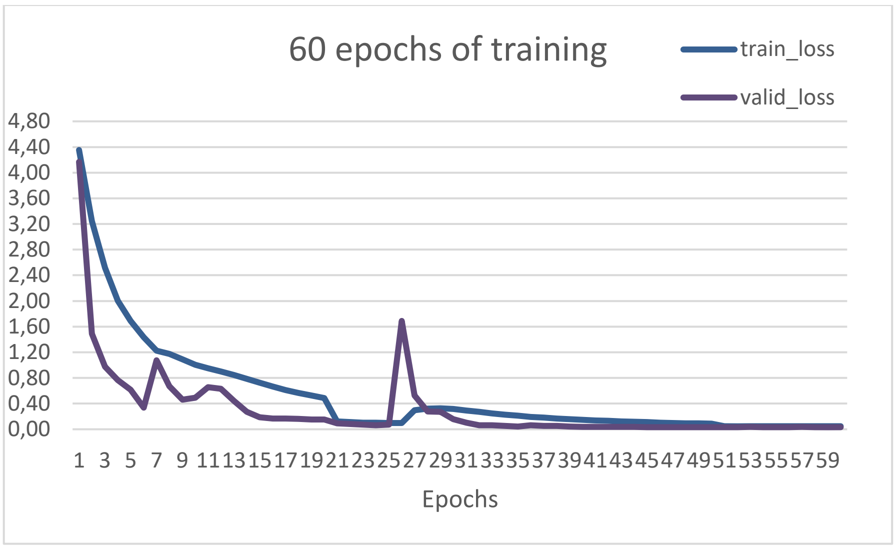


Figure 4.12: Loss metrics.

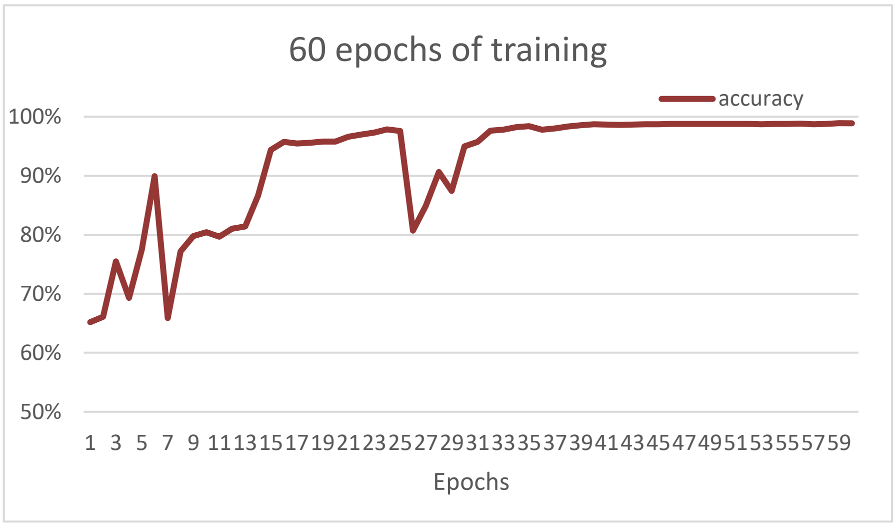
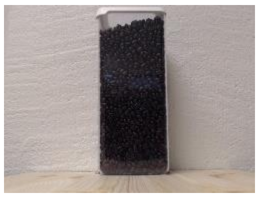
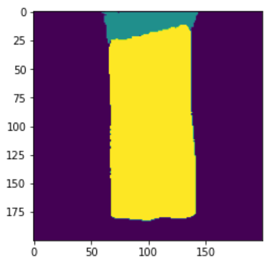
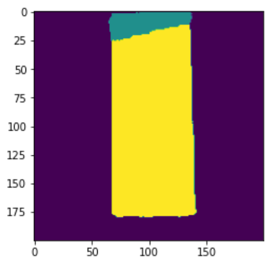
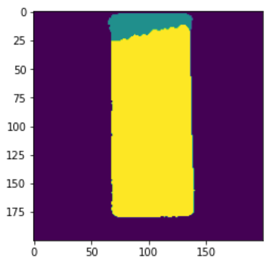
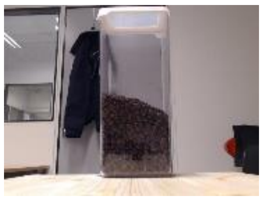
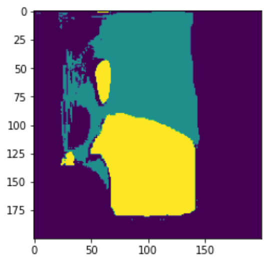
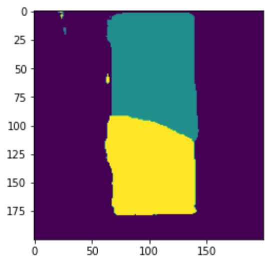
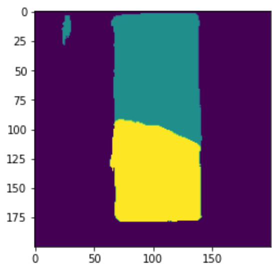


Figure 4.13: Model loss performance.

The amount of training needed varies from the complexity of the task. In Table 4.2 some examples of different situations are shown, where yellow is coffee, blue is container without coffee, and purple is everything else. In the first row, a situation with even lighting, little reflection and no noise in the background is shown. Here the machine learning model finds satisfactory results already after just 20 epochs. The problems appear when something like the background changes. This is shown in the second row when a dark jacket is introduced in the background. With only 20 epochs of training, the ML model struggles to differentiate the dark jacket from the container with coffee. At 40 epochs the prediction is much better, and at 100 epochs of training the prediction is almost perfect. The more complex the situation is, the more training the ML model need. It is important to have a variety of situations in the training data since the ML model only “knows” about what is present in the training data.

Table 4.2: Prediction result for images after n epochs of training.

Image	20 epochs	40 epochs	100 epochs
			
			

4.5.2 Prediction:

When the ML model training is finished and given an image of a coffee container it will create a tensor array containing data describing the mask layers corresponding to the given image. The resulting tensor array is plotted and gives the results shown in Table 4.2, this tensor is used to calculate the level in the coffee bean container using (2.1).

$$Coffee\ level\ [\%] = \frac{Coffee\ beans\ (Yellow)}{(Empty\ Container\ (Blue)\ +\ Coffee\ beans(Yellow))} \tag{4.1}$$

5 Traditional image processing

Traditional image processing is the use of digital computer process images through an algorithm. There is different type of images such as binary image, Black and white image, 8-bit color format and 16-bit color format. An image is defined as a two-dimensional array, that are arranged in rows and columns. A digital image has a finite number of pixels, each of which pixel have a specified value at a particular location in the image. Traditional image processing mainly includes the process of importing the image analyzing and manipulating images and a result where the resulting image can be an altered image. Advantages of traditional image processing is that it can often solve a given problem with fewer lines of code than machine learning [53].

OpenCV is an open-source library for machine learning and computer vision, and it is an abbreviation for “Open-Source Computer Vision Library”. OpenCV is a library that makes a common infrastructure for computer vision and in commercial products the use of machine perception increases [54].

The OpenCV algorithm is widely used around the world, examples of what it can be used for are the following [54]:

1. Recognize faces.
2. Classify movements of people.
3. Recognize objects.
4. Track moving parts.

OpenCV supports the programming languages C ++, Python, Java and Matlab. The operating systems that support OpenCV are Windows, Linux, Mac OS, and Android.

For further details, see the OpenCV Edge Detection website [54].

5.1 Binary Threshold Vs Canny Edge Detection

Binary Threshold and Canny Edge Detection (CED) are two functions in OpenCV library that can be used for processing different images.

5.1.1 Binary Threshold

With the function “cv.threshold” in OpenCV is it possible to perform different type of threshold operation. Which type of threshold is defined as the fourth parameter in the “cv.threshold” function. To read more about the different parameters for “cv.threshold” and all the types of thresholding available in OpenCV, visit OpenCV docs [55].

With the simplest thresholding each pixel will get applied the same threshold value to differentiate the pixels of interest. OpenCV performs thresholds operations on image through the function “cv.threshold”, it will perform a comparison of each and every pixel in the frame against the set value (threshold). A value will be set on each pixel if they are above or below the threshold range to identify them i.e., any value from 0 to 255 that best suited for the project [56].

$$dst(x,y) = f(x) = \begin{cases} 0, & if\ src(x,y) > thresh \\ 255, & otherwise \end{cases} \tag{5.1}$$

Equation (5.1) explains the “THRESH_BINARY_INV” algorithm [57]. If the intensity of the pixel src(x,y) is greater than that given threshold value it will set the value to black or grayscale value 0. Otherwise, it set the grayscale value of the pixel to 255 or white. The variation of intensity between the object and background pixels, this will separate out objects from the background which is the goal to analyze.

5.1.2 Canny Edge Detection

CED is an algorithm that can be used to detect edges in an image. Figure 5.4 shows what an output from CED can look like. CED is a multistage algorithm with the following steps:

1. Noise reduction with a Gaussian filter
2. Find intensity gradient of image.
3. Non-maximum Suppression
4. Hysteresis thresholding

This algorithm is built in to the OpenCV library through the “.Canny()” function [58]. Gaussian filtering smooths the image to remove noise. Finding the intensity gradient compares a pixel’s value against its neighboring pixels and outputs a new image where the larger difference in intensity equates to a higher pixel value [58]. Non-maximum suppression takes the output from intensity gradient, which contains all the edges, but the edges are of varying width. The goal of non-maximum suppression is to make these edges thinner [58]. Hysteresis thresholding has a max and min value it compares against the edges intensity gradient to determine if it is an edge or not [58]. If an edge is between these min/max values, it is classified based on its connectivity [58]. If an edge is between min/max and connected to a point over the max value, it is classified as an edge [58]. This is visualized in Figure 5.1.

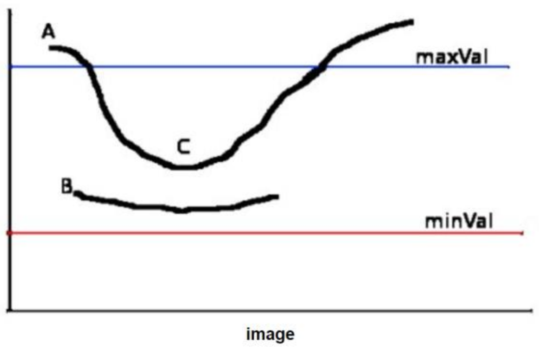


Figure 5.1 Hysteresis Thresholding [58].

5.1.3 Comparison

For this project, the goal was to measure the level in a coffee tank and the method used is Binary Threshold. The difference between Binary Threshold and CED is that with CED you find the edges of the container and the edge at the level of the coffee beans in the container. To find the edges of the coffee contents in the tank the “cv.canny()” functions goes through each stage inside a multi-stage algorithm that “cv.canny()” function use [58].
For Binary Threshold, all pixels are compared to a certain threshold value and for those below or above the threshold value, they are set to two different colors (black and white). Once all pixels have been checked up against the threshold value the function returns a new threshold image, where all the coffee is separated with another color. Binary Threshold was selected because it is easy to implement, and it works well under the right conditions.

Figure 5.2 Shows a flowchart of how Binary Threshold Inverted and Canny Edge detection works.

Figure 5.3 and Figure 5.4 shows the results from both Binary Threshold and Canny Edge detection. These two figures show the difference between these two functions in OpenCV. The methods use different images therefor the coffee level is different for these two figures, and the level cannot be compared.

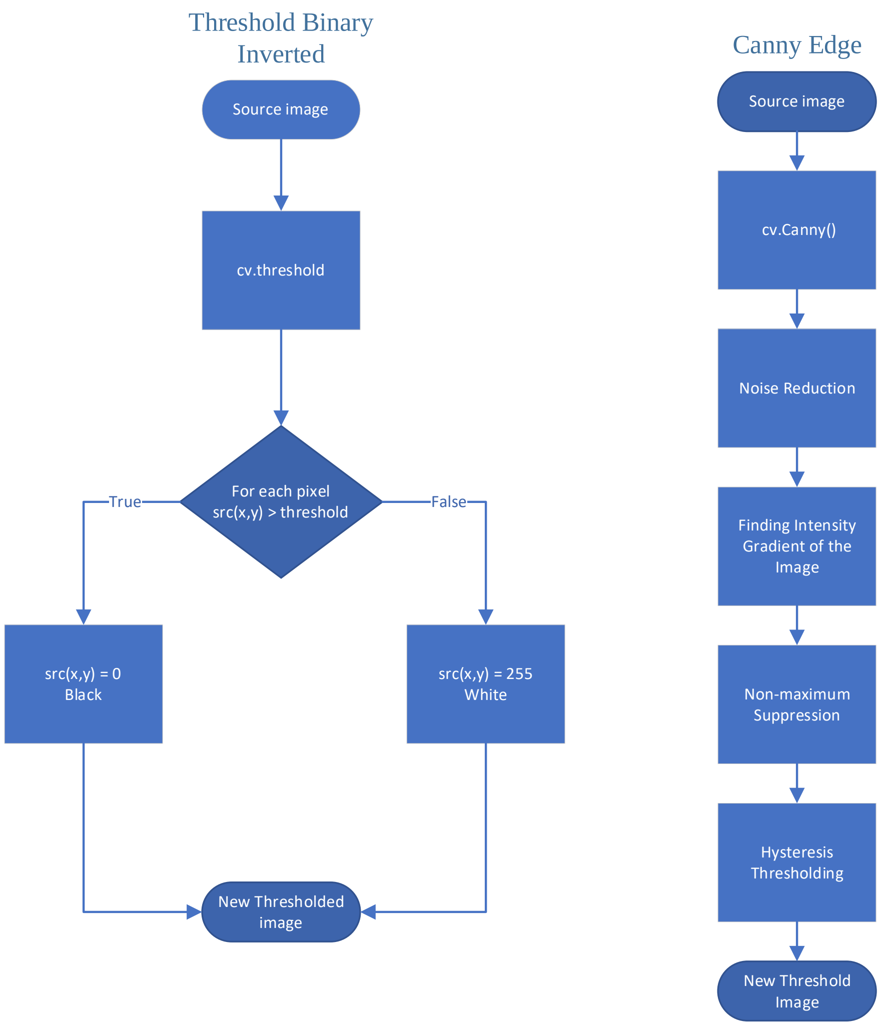


Figure 5.2 Flowchart shows Threshold Binary Inverted and CED.

In Figure 5.2 Threshold Binary Inverted have an input of an image which check every pixel's value or intensity against a predefined threshold value. If pixels intensity or source value is lower than the threshold value, the threshold function will set the intensity to 0 otherwise it will be 255.

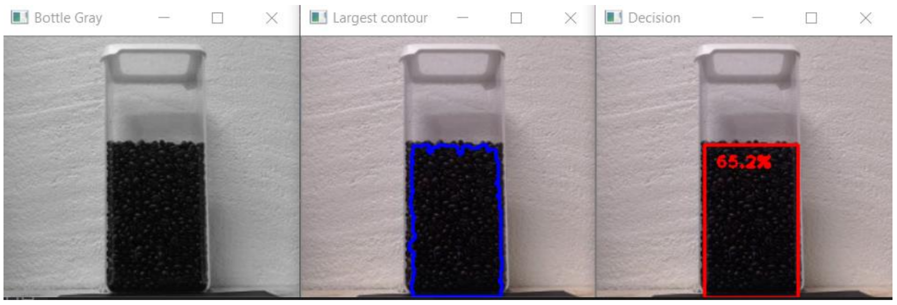


Figure 5.3 Results from Binary Threshold

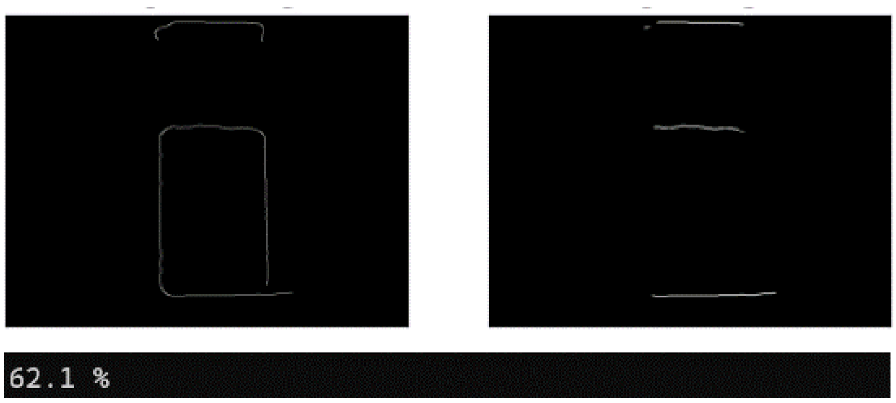


Figure 5.4 Results from CED

When attempting to estimate level from the output from CED, seen in Figure 5.4, there was issues with the lines in the image not being continuous. This resulted in inconsistencies when trying to detect the bottom edge, top of coffee edge and top of container edge. The aforementioned issues were more prevalent when the lighting conditions changed, and binary thresholding method proved more consistent.

5.2 Implementation

Initial experimentation found that CED was too sensitive to change in light conditions compared to binary thresholding. Due to this binary threshold was chosen as the preferred method. The resulting implementation is based on a blogpost from agmanic.com [59]. From this the software has been modified slightly to suit the problem in question, resulting in the steps seen in Figure 5.5.

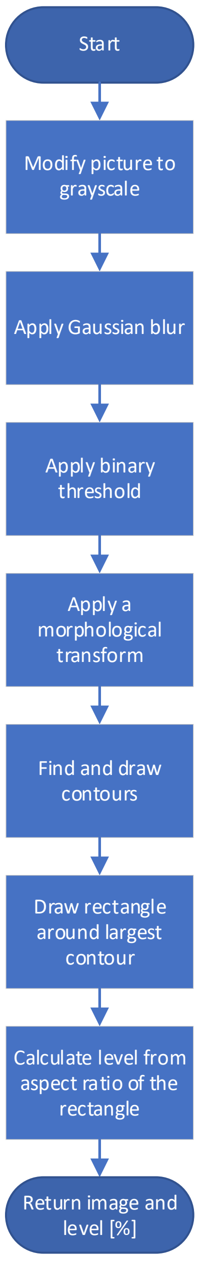

Start
Modify picture to grayscale
Apply Gaussian blur
Apply binary threshold
Apply a morphological transform
Find and draw contours
Draw rectangle around largest contour
Calculate level from aspect ratio of the rectangle
Return image and level [%]

Figure 5.5: Overview of steps in the traditional image processing method.

Figure 5.5 shows the steps used in the level estimation engine on the traditional image processing side. The first step is to convert the picture into grayscale format, this is done by taking an average of the Red Green Blue (RGB) values.

Step 2 is to apply a Gaussian blur. This is used to reduce the noise level in the image [60]. Gaussian filter can be considered a low-pass filter that removes insignificant details in an image [60].

As covered in section 5.1.1 step 3 is to apply the binary threshold. This results in an image with only completely black or completely white pixels.

Step 4 applies a morphological transformation called “opening” [61]. It is useful for removing noise in an image. Opening is a combination of the morphological transformations Erosion and Dilation, doing Erosion then Dilation [61]. Erosion is done by putting a kernel on each pixel of the image [61]. If not all the pixels in the kernel are the foreground color(white), it is set to the background color(black) [61]. Dilation is the exact opposite of erosion.

After the morphological transformation is applied, the next step (step #5) is to find contours in the image. *“Contours can be defined as a set of points along the boundary having the same color or intensity”* [62].

The contours get saved as a set of points defining the contour. Step 6 sends the points of the largest contour to the “cv.boundingRect” function. This calculates a rectangle that fits around the contour.

The last step is to calculate the level from the dimensions of the bounding rectangle. The algorithm for calculating the level makes two assumptions. The first assumption is that the bounding rectangle will always have the same width, that being the width of the container, the second assumption is that the container is always the same width to height ratio. The first upper two rows of Figure 5.6 shows the error due to the assumptions not being true. Equation (5.2) shows the two formulas needed for calculating the level in percent from the bounding rectangle. As previously mentioned, if the width of bounding rectangle changes the level estimate will no longer be correct.

$$container\ height\ in\ pixels[px] = \frac{width\ of\ bounding\ rectangle[px]}{\frac{container\ width[cm]}{container\ height[cm]}}$$
$$coffee\ level\ [\%] = \frac{height\ of\ bounding\ rectangle[px]}{container\ height\ in\ pixels[px]} * 100\% \tag{5.2}$$

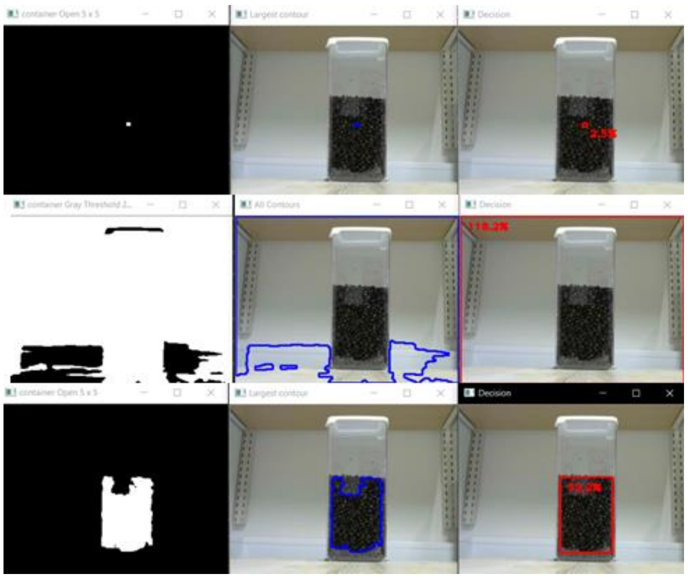


Figure 5.6: First column is output from binary thresholding, step 3. Second column is drawing contours overlayed over the original image, step 5. Third column is rectangle drawn around the largest contour, step 6. First row shows too low threshold value. Second row shows too high threshold value, bottom row shows an almost correct threshold value.

6 Results

This chapter contains the results from predictions with fastai and OpenCV in different scenarios. The first subchapter contains results from ideal scenarios, the second subchapter shows challenging scenarios, the third subchapter shows a comparison of time usage for predictions, and the fourth subchapter shows the results from a test of the projects repeatability.

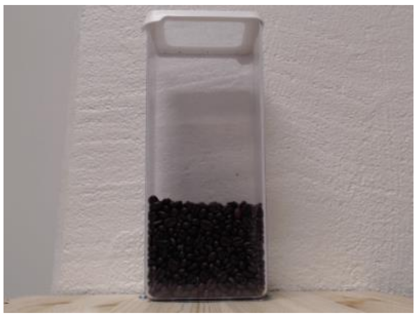
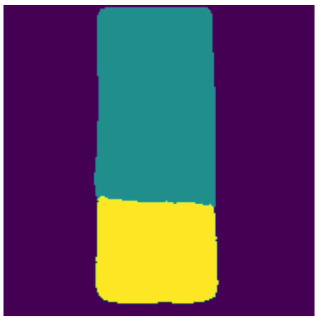
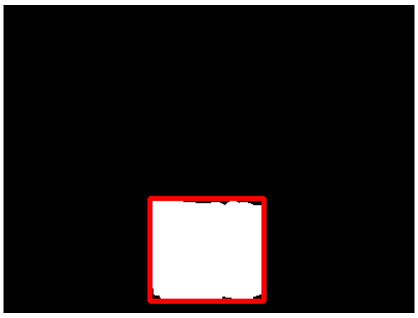
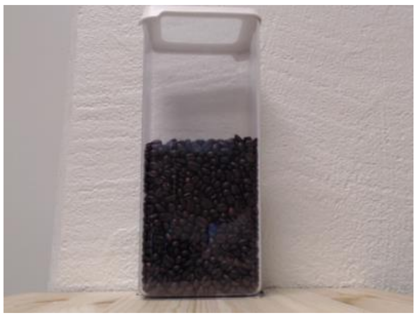
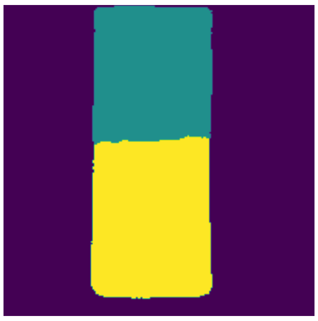
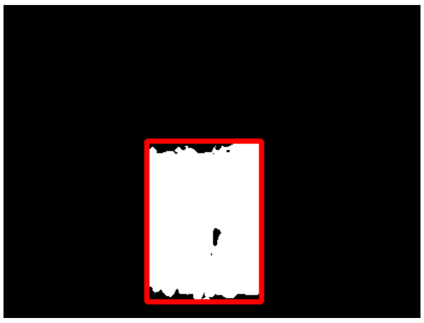
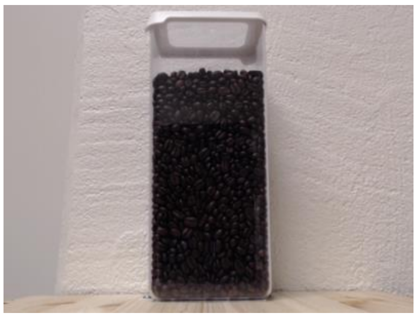
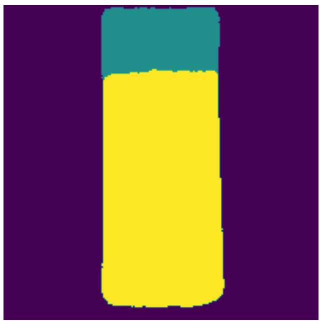
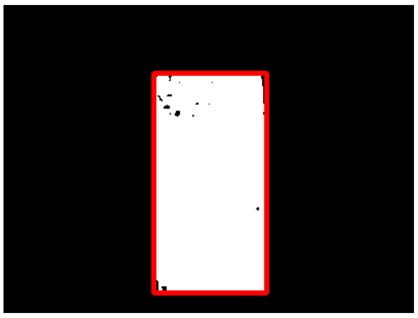
6.1 Ideal scenario

Table 6.1 compares the predicted level from fastai and OpenCV with a manual measurement. To get a representable comparison, only images that work with both models was chosen for this comparison. The average of all the results shows that fastai has an error margin of 0.7%, while OpenCV has an average error of around 3.0%. The models were also tested at 0% coffee level, but the level algorithm failed at both models. Table 6.2 shows some of the masks generated in the making of Table 6.1.

Table 6.1: Comparison of fastai and OpenCV against a manual measured Reference. Results marked in bold are shown in Table 6.2.

Reference [%]	Fastai [%]	Error [%]	OpenCV [%]	Error [%]
15,2	14,7	0,5	14,6	0,6
25,2	26,4	1,2	26,7	1,5
33,7	**34,3**	**0,6**	**37,9**	**4,2**
44,3	44,3	0,0	45,8	1,5
54,3	**54,0**	**0,3**	**59,0**	**4,7**
62,1	62,1	0,0	65,1	3,0
72,0	71,0	1,0	74,3	2,3
79,4	**78,8**	**0,6**	**82,0**	**2,6**
7,9	86,5	1,4	91,9	4,0
97,2	96,3	0,9	102,3	5,1
Mean error [%]:		0,7 %		3,0 %

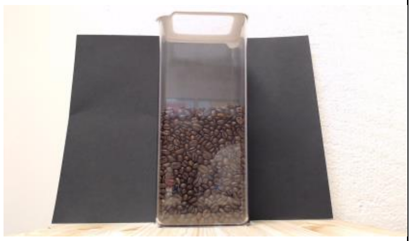
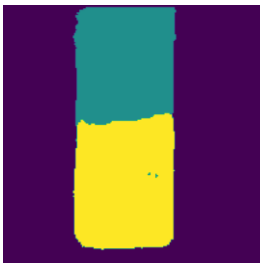
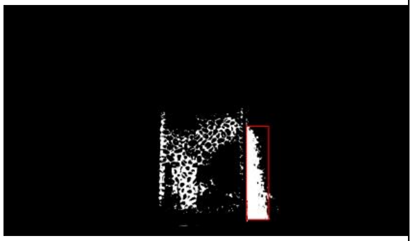
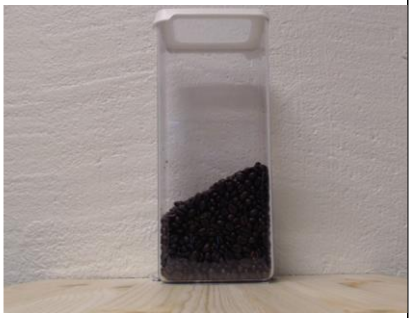
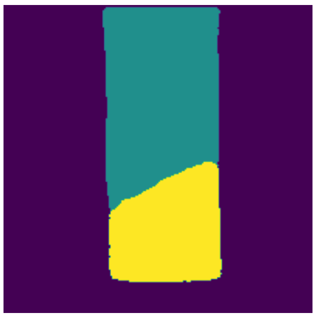
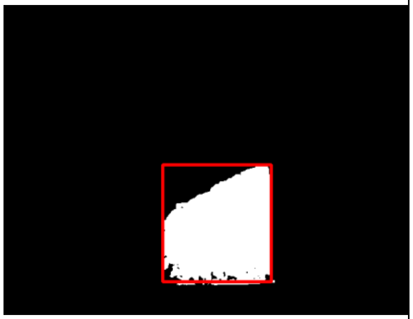
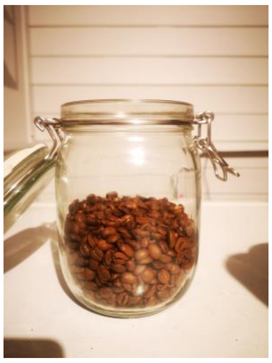
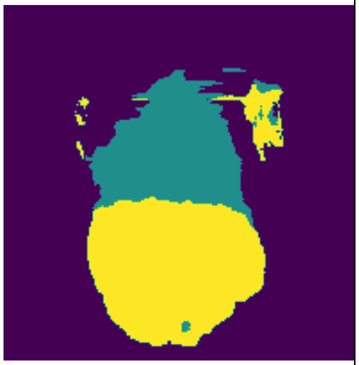
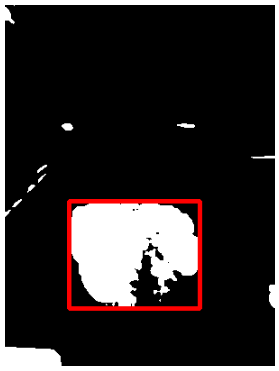
Table 6.2: Predicted mask and level from fastai and OpenCV with ideal scenarios

Input image	Fastai predicted mask	OpenCV predicted mask
 Measured: 33,7%	 Predicted: 34.3%	 Predicted: 37.9%
 Measured: 54.3%	 Predicted: 54.0%	 Predicted: 59.0%
 Measured: 79.4%	 Predicted: 78.8%	 Predicted: 82.0%

6.2 Challenging scenarios

Table 6.3 show the result from a few challenging situations. The first picture is with black background, the second with slanted coffee level and the third is with a completely different container than what is used in the training data.

Table 6.3: Predicted masks and level from fastai and OpenCV with challenging scenarios

Input Image	Fastai predicted mask	OpenCV predicted mask
 Measured 54.9%	 Predicted: 52.9%	 Predicted: 174.4%
 Measured: 34.7%	 Predicted: 36.3%	 Predicted: 45.2%
 Measured: 40.5%	 Predicted: 55.7%	 Predicted: 34.5%

6.3 Time Usage

Time usage for a single level prediction, where Table 6.4 shows the time difference between fastai and OpenCV.

Table 6.4: Comparing time needed to calculate level with OpenCV and fastai.

	OpenCV	Fastai	Time difference
Average time	0.01885 [s]	6.353 [s]	337 X
Fastest time	0.01326 [s]	2.751 [s]	208 X
Slowest time	0.03009 [s]	11.666 [s]	388 X

6.4 Repeatability

To test the repeatability of the predictions, three experiments where preformed. All three experiments estimated the coffee level 10 times with no more than one changing variable for each experiment. These experiments were conducted on the rig mentioned in section 2.2.

1. Test repeatability in the models with the same image.
2. Test the importance of angle with a rotating container with the same coffee level.
3. Test the importance of the way the coffee stacks with the same coffee amount.

The first experiment performing a level estimation on the exact same image-file 10 times gave the same result on each prediction. This is true for both OpenCV and fastai.

The second experiment is shown in Table 6.5. These results include the prediction of both methods as well as the calculated average and standard deviation and come from rotating the tank around an axis normal to the supporting surface, with the same amount of coffee in the tank on every picture. The stacking of the coffee was flat in this experiment. 0% means is empty and 100% is a full container.

Table 6.5: Showing the readings and calculation from rotating the tank $< 90°$ in relation to the camera.

Picture n	OpenCV [%]	Fastai [%]
1	59,4	56,2
2	60,6	57,2
3	71,7	59,2
4	64,6	59,0
5	66,3	57,7
6	64,7	53,9
7	58,3	53,4
8	56,8	54,4
9	62,8	54,9
10	63,5	58,2
Average [%]	59,64	56,41
Standard Deviation	4,59	2,04

The third experiment performed was to test how vulnerable the prediction is to the way that the coffee stacks in the container. Table 6.6 shows the results from ten predictions as well as the calculated average and standard deviation of these results.

Table 6.6:Showing the readings and calculation from refilling the tank for each picture with the same amount off coffee.

Picture n	OpenCV [%]	Fastai [%]
1	59,1	56,4
2	56,4	59,7
3	62,4	59,5
4	65,7	63,7
5	58,6	56,1
6	68,0	58,5
7	59,8	56,4
8	58,7	55,6
9	58,3	55,9
10	63,9	58,5
Average [%]	59,56	58,03
Standard Deviation	3,39	2,39

7 Discussion

This chapter contains the discussion of the results and other interesting observations form the project. The first subchapter covers ideal scenarios, the second subchapter shows challenging scenarios, the third subchapter covers the time used for predictions, and the fourth subchapter discusses the projects repeatability.

7.1 Ideal scenario

The ideal scenarios are a plain light-colored background with little disturbances, even lighting, and little to no reflection. In this situation both OpenCV and fastai detects the area with coffee beans correctly and can calculate the level with good accuracy. The mean error from ten different predictions gives fastai an error at 0,7% and OpenCV 3,0% as seen in chapter 6.1. The fact that both methods failed at 0 percent is due to the way the level is calculated in code. This can be improved with exception handling but was not done due to time constraints.

7.2 Challenging scenarios

Table 6.3 show some situations with more challenging scenarios. In the first row a black sheet was added to test if the models could differentiate black from the coffee beans. The results show that fastai still predicts almost perfectly, except for a small strip with a lot of reflection on top of the coffee beans. OpenCV struggles much more to separate the dark background from coffee. The dark background also caused the camera to set up the light sensitivity, which made the coffee too light to be within the threshold. For future work all automatic functions on the camera, like light sensitivity, should be turned off to secure better control over the images.

The second row shows an almost ideal scenario, but the coffee is slanted. Here OpenCV overestimate the level while fastai predicts quite closely. This is because the machine learning based approach counts all the marked pixels, while OpenCV draws a rectangle around the extreme points in the largest contour and find the height from a width to height ratio. OpenCV will therefore ignore indents and almost always overestimate the level, given that it finds all the coffee. A solution for this in future work can be to draw a polygon around the coffee instead of a rectangle.

The picture in the third row is of a completely different container than used in in the training data. One of the problems with this is that one can see the top surface of the coffee. This will affect the measurement as mentioned in chapter 2.2 .The coffee beans in the picture are also a much lighter color. Still, both models give a surprisingly good mask and a level within 15% of the correct level. The fastai model predicted the coffee to a surprising accuracy, only some areas around the lid were completely wrongly segmented. OpenCV only missed by 5%. This was just random because the ratio between the height and the width of the container is different from what is programmed and must be adapted to each container.

With OpenCV, it should be possible to create a much better solution than the one used here, but this requires better prior knowledge or better time than available during this project. The fact that the machine learning model is so robust and accurate compared to the traditional image processing method shows that machine learning can be a good tool even for someone with little to no prior experience.

A common factor in the experiments performed during this project is the sensitivity for reflection on the container. The reflections affect the results on both fastai and OpenCV, resulting in inaccurate level calculations. In fastai the reflections will in some cases be interpreted as the container instead of the coffee beans behind the reflection. In OpenCV the reflection brightness makes the algorithm avoid this area as it is brighter than the coffee beans not covered by a reflection.

7.2.1 Untrained scenarios

In fastai the model is trained using a set of reference images, see chapter 4.4.2. The fastai model can be retrained for new deviations or disturbances, this is done by adding images containing the new conditions to the model. Next the model is trained through several epochs and a new model is produced that can be exported to a new file for the model. The previously trained model should be preserved and is used as a starting point when a ML model is trained with new conditions. This is the same process as creating a new model using transfer learning on the ResNet-34 model, as described in chapter 4.3 [34].

In OpenCV threshold values are set for a specific scenario, see chapter 5.1.1. Because of OpenCV’s specific threshold limit values it is more vulnerable to changing light conditions and color disturbances than fastai. To Compensate for new deviations and disturbances it is necessary to manually set new threshold values according to a specific scenario. This is done programmatically meaning that for OpenCV to handle a new specific scenario a partial recoding is needed, this comes with the cost of lost accuracy in other scenarios.

The difference in effort needed when updating the methods for new conditions or deviations, shows the versatility and adaptability of the fastai training methods compared to the OpenCV’s methods of strict threshold settings. When compensating for disturbances fastai will retain all previous scenarios, while OpenCV is configured for using specific settings for each scenario.

7.2.2 Disturbances:

Performing image processing when the source image has disturbances or deviations from the reference images can cause the level prediction to fail or miscalculate the level. These disturbances and deviations are typically changes in the container geometry, challenging light conditions, color schemes, reflections, shadows, or new structures in the background.

7.3 Time usage

An experiment was done where 40 level predictions was run with both OpenCV and fastai to compare the time usage for each prediction. The experiment used a collection of randomly selected images and gave the results shown in in Table 6.4. The results show that OpenCV is impressively quick when performing the level measurement with an average time that is 337 times faster than fastai. When Looking at the fastest time this difference is reduced to OpenCV being 208 times faster than fastai. This can be explained by the different the way the image processing is done, fastai uses complex mathematic to calculate and compare pixel gradients, while OpenCV converts the image to greyscale and use simpler mathematics to find a binary threshold. Tracking the level in a container with coffee beans is not a high priority task as the level changes in a slow rate. This means that the relatively slow prediction times does not affect this project or an implementation to the AI barista robot mentioned in chapter 1. The coffee bean level is only changing when a cup of coffee is brewed or when the container is refilled. As the level only needs to be calculated when the level changes, the real time demand for the solution is the minimum amount of time it would take to brew a cup of coffee. The project solution meets the real time requirement, as making a cup of coffee take about the same time as performing the prediction.

7.4 Repeatability

Experiments testing the predictions repeatability were performed with both OpenCV and fastai, the results from the experiments are shown in chapter 6.4. This subchapter discusses and compares the results from both OpenCV and fastai.

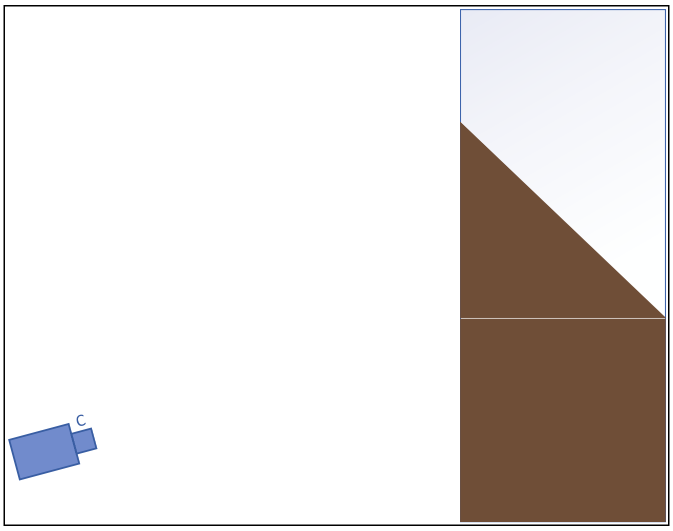


Figure 7.1: Coffee stacking that the models are vulnerable to error with.

7.4.1 Fastai

When referring to Table 6.5 and Table 6.6 in this subchapter, the authors refer to the fastai column.

The results from Table 6.4 show that fastai is more affected by the way the coffee stacks in the container than the rotating experiment. The ML model is less affected by rotation because it calculates level from the relationship between coffee and empty container, not the absolute amount. This means that the level will be calculated accurately, presuming the ML model recognizes the classes correctly.

In Table 6.6 one can see a higher standard deviation, meaning that the repeatability is worse in this experiment. This can be due to the way the coffee stacks when filling the container. If the coffee stacks in the same fashion as shown in Figure 7.1. the ML model will predict a much higher level than what it would if the level laid flat in the container. This is due to the level prediction algorithm, that effectively assumes that measured area correlates to total volume.

In future work one could try a two-camera setup with the cameras opposed 90degrees, or a stereo vision setup where two cameras are placed on a horizontal line with a set distance between them and angled towards the same object [63]. In both methods the two predicted areas could be combined into a more robust volume assumption, with the stereo vision it is also possible to triangulate the distance to the object and use this to increase the measurement accuracy.

7.4.2 OpenCV

When referring to Table 6.5 and Table 6.6 in this subchapter, it is referred to the OpenCV column.

The results in Table 6.5 shows that OpenCV is more affected by rotating than refilling the container. This is most likely because OpenCV uses the width to height ratio of the container to calculate the level. By rotating the container, one changes the width to height ratio and therefore gives the calculation an error. There is not a big difference between the average results in Table 6.5 and Table 6.6.

7.4.3 Comparison

When comparing the project from chapter 6.4, looking at both variation in the containers rotation and looking at how the coffee beans are dispersed, one can see how irregularities in the environment affect the accuracy of the predictions.

Looking at changes in the container's rotation, the standard deviation of OpenCV is more than doubled compared to the standard deviation of fastai. This can be explained by the methods used to calculate the level, where OpenCV uses the ratio between width and height, whilst fastai calculates the ratio between the segmented areas defined as coffee and container.

The experiment also tested how the two methods handles differences in how the coffee beans are dispersed in the container. In this experiment the difference in standard deviation was smaller, but still significant for prediction results. The standard deviation of OpenCV is about 40% higher than the standard deviation of fastai. This can be due to the OpenCV approach drawing a rectangle around the contours which will effectively remove some resolution.

From these two experiments it is shown that OpenCV is more sensitive to the container's rotation than how the coffee beans are dispersed. The opposite is true for fastai which is more sensitive for changes in how the coffee beans are dispersed than how the container is rotated.

An interesting observation from these experiments is that the different models have each their weaknesses and strengths. As a note, it is important to pay attention details like, camera placement, container rotation, how the medium is dispersed in the container, and general disturbances in the environment like reflections from the surroundings.

8 Conclusion

The purpose of this project was to identify the usefulness of machine learning with computer vison in comparison with traditional image processing. Based on the analysis conveyed it can be concluded that computer vision with machine learning is superior to traditional image processing in terms of accuracy, versatility, and user-friendliness. OpenCV is still a good choice for level prediction, being faster and less resource demanding than fastai, although less accurate. This assuming an ideal environment with even lighting and without disturbances. For the project for which the models are made, both are more than good enough to perform the task of finding the coffee level in the barista.

In future work, it should be compensated for lens and perspective distortion and it should also be considered to mount a second camera to be able to estimate the level more accurately. In traditional image processing one could change the rectangle around the extreme points, to a polygon. This would perhaps make the model more robust against slanted coffee level.

Due to the time constraints of the project the website did not implement phone functionality, this would be beneficial in future work. There are also some problems with https certificates, this was not solved within the timeframe given.

The accuracy and repeatability are found to be acceptable for the implementation in a coffee machine, but in another application, there may be a higher demand for correcting distortion and the robustness against stacking errors.

9 References

[1] J. P. Bentley, "Principles of measurement systems," 2005.

[2] I. Goodfellow, Y. Bengio and A. Courvil, Deep learning, MIT Press Cambridge, 2016.

[3] L. E. Finholt, "Kafferobot," Bouvet, 23 April 2021. [Online]. Available: https://www.bouvet.no/bouvet-deler/kafferobot. [Accessed 11 May 2021].

[4] Mozilla, "HTTP request methods," 4 December 2020. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods. [Accessed 18 May 2021].

[5] OpenCv, "Camera calibration With OpenCV," [Online]. Available: https://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/camera_calibration.html. [Accessed 04 May 2021].

[6] M. H. Jones, "Calibration Chekerboard Collection," 18 April 2018. [Online]. Available: https://markhedleyjones.com/projects/calibration-checkerboard-collection. [Accessed 04 May 2021].

[7] Google Research, "What is Colaboratory?," Google Research, [Online]. Available: https://colab.research.google.com/notebooks/intro.ipynb. [Accessed 2 2021].

[8] R. B. Hegde, K. Prasad, H. Hebbar and B. M. K. Singh, "Comparison of traditional image processing and deep learning approaches for classification of white blood cells in peripheral blood smear images," Biocybernetics and Biomedical Engineering, 2019.

[9] E. Døble, S. Haugseter, C. Mikkelsen and J. Sneisen, "Project Repo LevelApi," 19 March 2021. [Online]. Available: https://github.com/IA6-5-21/LevelAPI. [Accessed 19 March 2021].

[10] E. Døble, S. Haugseter, C. Mikkelsen and J. Sneisen, "Project Repo Website," 6 March 2021. [Online]. Available: https://github.com/IA6-5-21/CoffeeFinder. [Accessed 2021 March 2021].

[11] E. Døble, S. Haugseter, C. Mikkelsen and J. Sneisen, "Project Repo Training Code," February 2021. [Online]. Available: https://github.com/IA6-5-21/ModelTraining. [Accessed 11 May 2021].

[12] Microsoft, "AppService overview," 7 June 2020. [Online]. Available: https://docs.microsoft.com/en-us/azure/app-service/overview. [Accessed March 2021].

[13] The World Wide Web Consortium, "HTML 5.2," [Online]. Available: https://www.w3.org/TR/html52/. [Accessed 04 2021].

[14] The World Wide Web Consortium, "HTML Media Capture," [Online]. Available: https://w3c.github.io/html-media-capture/. [Accessed 04 2021].

[15] Ruderman Jesse, "Same-origin policy," mozilla, 2021. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy. [Accessed 05 2021].

[16] W3Schools, "Botstrap Theme 'company'," [Online]. Available: https://www.w3schools.com/bootstrap/bootstrap_theme_company.asp. [Accessed 19 February 2021].

[17] Docker, [Online]. Available: https://www.docker.com/resources/what-container. [Accessed April 2021].

[18] FastApi, "FastApi Docs," [Online]. Available: https://fastapi.tiangolo.com/. [Accessed April 2021].

[19] Mozilla, "JSON," 5 May 2021. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON. [Accessed 18 May 2021].

[20] Mozilla, "Base64," 16 March 2021. [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/Base64. [Accessed 18 May 2021].

[21] E. Døble, S. Haugseter, C. Mikkelsen and J. Sneisen, 04 2021. [Online]. Available: https://52.142.127.98/docs. [Accessed 04 2021].

[22] Microsoft, "Azure Container Instances," [Online]. Available: https://azure.microsoft.com/en-us/services/container-instances/. [Accessed 20 May 2021].

[23] sas, "machine mearning," [Online]. Available: https://www.sas.com/en_us/insights/analytics/machine-learning.html. [Accessed 05 2021].

[24] J. Brownlee, "machinelearningmastery," 2019. [Online]. Available: https://machinelearningmastery.com/what-is-computer-vision/. [Accessed 25 3 2021].

[25] microsoft, "what is a machine learning model," 2019. [Online]. Available: https://docs.microsoft.com/en-us/windows/ai/windows-ml/what-is-a-machine-learning-model. [Accessed 05 2021].

[26] thinkautomation, "what is image classification in deep learning?," [Online]. Available: https://www.thinkautomation.com/eli5/eli5-what-is-image-classification-in-deep-learning/. [Accessed 05 2021].

[27] Cogito Tech LLC, "meduim," 28 11 2019. [Online]. Available: https://medium.com/cogitotech/what-is-the-difference-between-image-segmentation-and-classification-in-image-processing-303d1f660626. [Accessed 03 2021].

[28] labelbox, "Image Segmentation 101," [Online]. Available: https://labelbox.com/image-segmentation-overview. [Accessed 05 2021].

[29] "Residual neural network," 2021. [Online]. Available: https://en.wikipedia.org/wiki/Residual_neural_network. [Accessed 20 04 2021].

[30] T. Gupta, "Deep Learning: Feedforward Neural Network," https://towardsdatascience.com/, 05 01 2015. [Online]. Available: https://towardsdatascience.com/deep-learning-feedforward-neural-network-26a6705dbdc7. [Accessed 05 2021].

[31] Y. Minghao , L. Xiu, Z. Yongbing and W. Shiqi , "arixiv," 16 04 2019. [Online]. Available: https://arxiv.org/pdf/1904.07568.pdf. [Accessed 04 2021].

[32] C.-F. Wang, "The Vanishing Gradient Problem," 01 2019. [Online]. Available: https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484. [Accessed 05 2021].

[33] "imagenet," 02 2021. [Online]. Available: https://www.image-net.org/. [Accessed 05 2021].

[34] Sasank Chilamkurthy, "Transfer learning for computer vision tutorial," PyTorch, 2021. [Online]. Available: https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html. [Accessed 5 2021].

[35] . J. Brownlee , "Overfitting and Underfitting With Machine Learning Algorithms," 03 2016. [Online]. Available: https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/. [Accessed 05 2021].

[36] PyTorch, "TORCHVISION.MODELS," PyTorch, 2017. [Online]. Available: https://pytorch.org/vision/stable/models.html. [Accessed 04 2021].

[37] fast.ai, "Practical Deep Learning for Coders," fast.ai, 2021. [Online]. Available: https://course.fast.ai/. [Accessed 03 2021].

[38] J. Howard and R. Thomas, "fast.ai - Making neural nets uncool again," 2021. [Online]. Available: https://www.fast.ai/. [Accessed 01 2021].

[39] Facebook, "PyTorch," Facebook, 2021. [Online]. Available: https://ai.facebook.com/tools/pytorch/. [Accessed 05 2021].

[40] J. Brownlee, "A Gentle Introduction to Tensors for Machine Learning with NumPy," Machine Learning Mastery Pty. Ltd, 14 02 2021. [Online]. Available: https://machinelearningmastery.com/introduction-to-tensors-for-machine-learning/. [Accessed 05 2021].

[41] fast.ai, "Vision data," fast.ai, 2021. [Online]. Available: https://docs.fast.ai/vision.data.html. [Accessed 04 2021].

[42] fast.ai, "DataLoader helpers," fast.ai, 2021. [Online]. Available: https://docs.fast.ai/data.load.html. [Accessed 05 2021].

[43] PyTorch, "TORCHVISION.TRANSFORMS," PyTorch, 2017. [Online]. Available: https://pytorch.org/vision/stable/transforms.html. [Accessed 03 2021].

[44] fast.ai, "Learner for the vision applications," fast.ai, 2021. [Online]. Available: https://docs.fast.ai/vision.learner.html. [Accessed 04 2021].

[45] O. Ronneberger, P. Fischer and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," *Medical Image Computing and Computer-Assisted Intervention (MICCAI),* vol. 9351, pp. 234--241, 2015.

[46] V. Kumar, "PyTorch Autograd - Understanding the haart of PyTorch's magic," Towards data science, 2019. [Online]. Available: https://towardsdatascience.com/pytorch-autograd-understanding-the-heart-of-pytorchs-magic-2686cd94ec95. [Accessed 03 2021].

[47] PyTorch, "A gentle introduction to torch.autograd," 2021. [Online]. Available: https://pytorch.org/tutorials/beginner/blitz/autograd_tutorial.html. [Accessed 04 2021].

[48] A. Gad, "A Comprehensive Guide to the Backpropagation Algorithm in Neural Networks," Neptuneblog, 03 2021. [Online]. Available: https://neptune.ai/blog/backpropagation-algorithm-in-neural-networks-guide. [Accessed 04 2021].

[49] A. Opidi, "PyTorch Loss Functions: The Ultimate Guide," Neptune , 2021. [Online]. Available: https://neptune.ai/blog/pytorch-loss-functions. [Accessed 04 2021].

[50] D. Yue, Y. He and Y. Li, "Piston Error Measurement for Segmented Telescopes with an Artificial Neural Network," College of Science, Changchun University of Science and Technology, Changchun 130022, China, 12 05 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/10/3364/htm. [Accessed 18 05 2021].

[51] "Overfitting and Underfitting," fast.ai, [Online]. Available: https://www.fastaireference.com/overfitting. [Accessed 05 2021].

[52] Fast.ai, "Tracking callbacks," Fast.ai, 2021. [Online]. Available: https://docs.fast.ai/callback.tracker.html. [Accessed 04 2021].

[53] HCL Technologies, "Decoding the dichotomy: Traditional Image Processing vs. Deep Learning," p. 6.

[54] OpenCV, "OpenCV/About," 2021. [Online]. Available: https://opencv.org/about/. [Accessed 17 04 2021].

[55] OpenCV, "Image Thresholding," [Online]. Available: https://docs.opencv.org/master/d7/d4d/tutorial_py_thresholding.html. [Accessed 1 May 2021].

[56] OpenCV, "Basic Thresholding Operations," [Online]. Available: https://docs.opencv.org/4.5.2/db/d8e/tutorial_threshold.html.

[57] OpenCV, "Miscellaneous Image Transformations," [Online]. Available: https://docs.opencv.org/3.4/d7/d1b/group__imgproc__misc.html#ggaa9e58d2860d4afa658ef70a9b1115576a19120b1a11d8067576cc24f4d2f03754. [Accessed 09 05 2021].

[58] OpenCV, "Canny Edge Detection," [Online]. Available: https://docs.opencv.org/master/da/d22/tutorial_py_canny.html. [Accessed 5 May 2021].

[59] P. McLaughlin, "Agmanic Vision," 7 August 2019. [Online]. Available: https://agmanic.com/detect-bottle-fill-level-with-50-lines-of-python/. [Accessed February 2021].

[60] M. Siddharth, L. Hao and H. Jiabo, "Machine Learning for Subsurface Characterization," in *Machine Learning for Subsurface Characterization*, Gulf Professional Publishing, 2020, pp. 39-64.

[61] OpenCV, "Morphological Transdormations," [Online]. Available: https://docs.opencv.org/master/d9/d61/tutorial_py_morphological_ops.html. [Accessed 06 May 2021].

[62] OpenCV, "Opencv Contours," [Online]. Available: https://docs.opencv.org/master/d4/d73/tutorial_py_contours_begin.html. [Accessed 22 May 2021].

[63] I. Takashi, U. Yoshiharu and T. Yutaka, "Height Measurement System Using Stereo Vision," Nihon University, Funabashi-City, 2003.

Appendices

Appendix A Prosjektbeskrivelse.pdf

Universitetet i Sørøst-Norge

Fakultet for Teknologi, Naturvitenskap og Maritime fag

PRH612 Bacheloroppgave

Studieretning: Informatikk og automatisering

Prosjektgruppe: IA6-5-21

Tittel:

Veileder: Ole Magnus Brastein

Bi-Veileder: Nils-Olav Skeie

Samarbeidspartner: Bouvet

Prosjektets bakgrunn:

Prosjektet er en del av en robotisert kaffemaskin fra Bouvet hvor prosjektpartneren ønsker å sammenligne nivåmåling med en trent maskinlæringsmodell mot tradisjonell bildebehandling. Dette skal skje i en webapplikasjon. Grunnet Coronasituasjonen i 2020/21 er kafferoboten utlignelig og det må derfor lages en egen testrigg for å kunne trene og teste programmet.
Mer tradisjonelle målemetoder som dette kan sammenlignes med kan være:

- Ultralyd
- Radar
- Vekt

Fordelene med måling ved hjelp av «computer vision» vil være at det er lett å ettermontere i en gjennomsiktig tank og at det ikke berører mediet eller tanken.

Målbeskrivelse for prosjektet:

Målet med prosjektet er å bruke maskinlæringsmodell som skal sammenlignes med nivåmåling basert på tradisjonell bildebehandling.
Metodene skal implementeres og sammenlignes i en webapplikasjon, hvor et kamera tar bilder av en kaffebønne-tank og sender bildene til prosessering for å estimere nivå i en beholder. En stor del av oppgaven vil være å trene og tilpasse maskinlæringsmodellen der det må planlegges og utføres eksperimenter for å samle inn bilder som kan brukes til treningsdata.

Oppgaver

1. Bygge eksperimentell rigg med tank og webkamera. Kamera og tank skal være fastmontert på en plate, noe som sikrer repeterbarhet uavhengig av lokasjonen til riggen. Det skal testes om det er behov for en fast bakgrunn i bildene.
2. Prøve ut forskjellige maskinlæringsalgoritmer og sammenlikne

- **2.1.** Undersøke aktuelle rammeverk – fast.ai
- **2.2.** Teste brukervennlighet
- **2.3.** Sammenligne resulterende modeller og brukervennligheten til disse.
- **2.4.** Velge den beste kombinasjonen av kvalitet vs. brukervennlighet.

3. Trene valgt maskinlæringsmodell
- **3.1.** Ta bildeserier med varierende nivå i tank/beholder til klassifisering.
- **3.2.** Eksperimentere med forskjellig «oppløsning». F.eks. 4 nivåpunkter mot 8 punkter.
- **3.3.** Verifisere at modellen gir tilfredsstillende nøyaktighet.

4. Lage nivåmåling basert på tradisjonell bildebehandling
- **4.1.** Undersøke aktuelle metoder for bildebehandling
- **4.2.** Lage algoritme for bildebehandling
- **4.3.** Verifisere at modellen gir tilfredsstillende nøyaktighet

5. Hoste maskinlæringsmodell og bildebehandlingsmodell i en skytjeneste (Azure).
- **5.1.** Finne egnet azure tjeneste som feks. Azure functions.
- **5.2.** Laste opp modell til azure.

6. Lage nettside for å vise sammenlikning av maskinlæringsmodell mot tradisjonell bildebehandling
- **6.1.** Grovskissere design av nettside.
- **6.2.** Velge teknologi. Avgjøre om nettsiden skal være responsiv.
- **6.3.** Finne egnet azure tjeneste for å hoste nettsiden. F.eks. Azure App Service.
  - **6.3.1.** Sørge for kommunikasjon mellom webside og modellene(Vnet?).
- **6.4.** Lage nettside iht. design.
- **6.5.** Testing

Eventuelt:

a. Repeter steg 2-4 for å telle kopper

b. Skrive forskningsartikkel
  - o Skrive abstract
  - o Skrive utvidet abstract
  - o Levere kladd av hel artikkel
  - o Lage manus

Signatur

Veiledere:

Ole M. Brastein
____
< Ole Magnus Brastein >

[illegible]
____
<Duyen My Tran>

____
< bouvet>

____
< bouvet >

Universitetet i Sørøst-Norge

Fakultet for Teknologi, Naturvitenskap og Maritime fag

bouvet

Studenter:

[illegible]
<Jørgen Sneisen>

Eirik Døble
<Eirik Døble>

[illegible]
<Sindre haugseter >

[illegible]
<Christian Mikkelsen>